Gary DeWard Brown

System/360 Operating System Job Control-Sprache

...en der Datenverarbeitung
...bourg Verlag München Wien
...iley & Sons GmbH Frankfurt

Verfahren der Datenverarbeitung

Herausgeber:
Klaus Becker-Berke und **Rudolf Herschel**

System / 360 Operating System Job Control-Sprache

von **Gary DeWard Brown**

Mit 3 Abbildungen, 4 Tabellen
und zahlreichen Job Control-Beispielen

1972
R. Oldenbourg Verlag München Wien
John Wiley & Sons GmbH Frankfurt

Die englischsprachige Originalausgabe **System / 360 Job Control Language**
ist erschienen im Verlag John Wiley & Sons, Inc., New York.
Copyright (C) 1970 by John Wiley & Sons, Inc. Alle Rechte vorbehalten.

Übersetzung: Peter Fiege, München.

© 1972 R. Oldenbourg München und
John Wiley & Sons GmbH, Frankfurt am Main

Das Werk ist urheberrechtlich geschützt. Die dadurch begründeten Rechte, insbesondere die der Übersetzung, des Nachdrucks, der Funksendung, der Wiedergabe auf photomechanischem oder ähnlichem Wege sowie der Speicherung und Auswertung in Datenverarbeitungsanlagen, bleiben, auch bei nur auszugsweiser Verwertung, vorbehalten. Werden mit schriftlicher Einwilligung des Verlages einzelne Vervielfältigungsstücke für gewerbliche Zwecke hergestellt, ist an den Verlag die nach § 54 Abs. 2 UG zu zahlende Vergütung zu entrichten, über deren Höhe der Verlag Auskunft gibt.

Satz: R. & J. Blank, München Druck: Graphische Anstalt E. Wartelsteiner, Garching-Hochbrück
Bindearbeiten: R. Oldenbourg Graphische Betriebe GmbH, München

ISBN: 3-486-33971-0

Inhalt

Vorwort 13

Die Parameter der Job Control-Sprache 14

1. Einführung 17

1.1 Die Rolle der JCL im System/360 17
1.2 Ziel und Zweck dieses Buches 17

2. Einführung in die JCL und das System/360 19

2.1 JCL-Karten 19
2.2 System/360: Aufbau und Begriffe 20
2.3 Betriebbsystem OS (Operating System) 20
2.3.1 OS/360-Systemarten 21
2.3.2 Multiprogramming 21
2.4 Jobs 25
2.4.1 Jobsteps 26
2.4.2 Compilation 27
2.4.3 Linkage Editor 27
2.4.4 Ausführung 28
2.4.5 Der Loader 29
2.5 Ein-/Ausgabe 30
2.5.1 Datenorganisation 30
2.5.2 Daten im Rechner 31
2.5.3 Zugriffstechniken 34
2.6 Datenspeicherung 35
2.6.1 Karten 35
2.6.2 Magnetbänder 35
2.6.3 Speicher mit direktem Zugriff 36
2.6.4 Datenträger 38
2.6.5 Die DD-Karte 38

3.	**JCL im Job**	40
3.1	Compile-Step	40
3.2	Linkage Editor-Step	42
3.3	Ausführungs-Step	44
3.4	Katalogisierte Verfahren	45
4.	**Formate und Regeln für die JCL-Karten**	47
4.1	Format der JCL-Anweisung	47
4.2	Parameter im Operandenfeld	48
4.3	Regeln für Parameter	49
4.4	Allgemeine Regeln	50
5.	**Beschreibung der JOB-Karte**	53
5.1	Der Jobname: jobname	54
5.2	Abrechnungsinformationen	55
5.3	Programmierername: name	56
5.4	Andrucken der JCL-Karten: MSGLEVEL	57
5.5	Jobpriorität: PRTY (nur MFT und MVT)	58
5.6	Systemnachrichten: MSGCLASS (nur MFT und MVT)	59
5.7	Jobklasse: CLASS (nur MFT und MVT)	60
5.8	HOLD-Job in der Eingabe-Queue: TYPRUN (nur MFT und MVT)	60
5.9	Zusammenfassung	61
6.	**Beschreibung der EXEC-Karte**	63
6.1	Der Jobstepname: stepname	64
6.2	Der Programmname: programm	64
6.2.1	Programme in der Systembibliothek	65
6.2.2	Programme in privaten Bibliotheken	65
6.2.3	Die JOBLIB DD-Karte	66
6.2.4	Die STEPLIB DD-Karte	67
6.2.5	Programme in temporären Bibliotheken	68
6.3	Der Name eines katalogisierten Verfahrens: prozedur	69
6.4	Schlüsselwortparameter	70
6.5	Parameterübergabe an Jobsteps: PARM	71
6.6	Abrechnungsinformationen für den Jobstep: ACCT	74
6.7	Dispatching-Priorität: DPRTY (nur MVT)	74

6.8 Dumps bei abnormaler Beendigung: SYSUDUMP, SYSABEND 76
6.9 Zusammenfassung 77

7. Parameter der JOB- und EXEC-Karten 79

7.1 Zeitbegrenzung: TIME (nur MVT) 79
7.1.1 Der Parameter TIME in der EXEC-Karte 80
7.1.2 Der Parameter TIME in der JOB-Karte 81
7.2 Bedingungen für das Überspringen von Jobsteps: COND . 81
7.2.1 Der Parameter COND in der EXEC-Karte 82
7.2.2 Der Parameter COND in der JOB-Karte 84
7.3 Rollout/Rollin-Angaben: ROLL (nur MVT) 84
7.3.1 Der Parameter ROLL in der EXEC-Karte 85
7.3.2 Der Parameter ROLL in der JOB-Karte 85
7.4 Regiongröße: REGION (nur MVT) 86
7.4.1 Der Parameter REGION in der EXEC-Karte 87
7.4.2 Der Parameter REGION in der JOB-Karte 88
7.5 Wiederanlaufangaben: RD, RESTART 88
7.5.1 Die Wiederanlaufangabe RD 89
7.5.2 Der Parameter RD in der EXEC-Karte 90
7.5.3 Der Parameter RD in der JOB-Karte 91
7.5.4 Abgabe eines Jobs für den Wiederanlauf: RESTART . . . 91
7.5.5 Wiederanlauf am Stepbeginn 91
7.5.6 Wiederanlauf von einem Checkpoint 93
7.6 Zusammenfassung 94

8. Beschreibung der DD-Karte 96

8.1 Data Control Block 97
8.2 Format der DD-Karte 98
8.3 Data Definition Name: ddname 99
8.4 Parameterrückbezug: rückbezug 100
8.5 Data Control Block-Angaben: DCB 100
8.5.1 Dateiorganisation: DSORG 101
8.5.2 Länge des logischen Satzes: LRECL 101
8.5.3 Länge des Blockes: BLKSIZE 101
8.5.4 Anzahl der Puffer: BUFNO 102
8.5.5 Satzformat: RECFM 102
8.5.6 DCB-Angaben aus dem Kennsatz 103
8.5.7 Übernahme von DCB-Angaben aus einer vorangegangenen DD-Karte oder Datei: rückbezug, name 103

8.6 Name der Datei: DSN 105
8.6.1 Temporäre Dateien 105
8.6.2 Nicht-temporäre Dateien 106
8.6.3 Andere Dateien 108
8.7 Ein-/Ausgabe-Einheit: UNIT 109
8.7.1 Hardware-Adresse 110
8.7.2 E/A-Geräte-Typ 110
8.7.3 E/A-Geräte-Gruppe 112
8.7.4 Einheiten-Affinität 113
8.7.5 Besondere Optionen 113
8.7.5.1 Paralleles Laden von Datenträgern 114
8.7.5.2 Verzögertes Laden eines Datenträgers: DEFER 114
8.7.5.3 Trennung von Dateien: SEP 115
8.8 Disposition der Datei: DISP 116
8.8.1 Status der Datei 116
8.8.1.1 Neue Dateien: NEW 117
8.8.1.2 Modifizierte Dateien: MOD 117
8.8.1.3 Bestehende Dateien: OLD 118
8.8.1.4 Gemeinsame Benutzung von Dateien: SHR 118
8.8.2 Normale Disposition 119
8.8.2.1 Weiterreichen von Dateien: PASS 120
8.8.2.2 Halten von Dateien: KEEP 121
8.8.2.3 Löschen von Dateien: DELETE 122
8.8.2.4 Katalogisieren von Dateien: CATLG 122
8.8.2.5 Entkatalogisieren von Dateien: UNCATLG 123
8.8.3 Abnormale Disposition 124
8.9 Parameter für den Datenträger: VOL 126
8.9.1 Anforderung für einen bestimmten Datenträger: SER, REF 128
8.9.2 Besondere Optionen 129
8.9.2.1 Datenträger vom Typ Private: PRIVATE 130
8.9.2.2 Halten von Private Datenträgern: RETAIN 131
8.9.3 Folgenummer 131
8.9.4 Anzahl 132
8.10 Zusammenfassung 134

9. Katalogisierte Verfahren 136

9.1 Verknüpfung von Dateien 136
9.2 Plazierung von DD-Karten 137
9.3 Katalogisierte Verfahren 142

9.3.1 Hinzufügung von Verfahren zur Verfahrensbibliothek . . 143
9.3.2 Testen von Verfahren 146
9.3.3 Symbolische Parameter 146
9.3.3.1 PROC-Anweisung 147
9.3.3.2 Kodierung symbolischer Parameter 147
9.3.3.3 Anwendung symbolischer Parameter 151
9.4 Zusammenfassung 153

10. DD-Karten für periphere E/A-Geräte 155

10.1 Dateien im Eingabestrom: * , DATA 155
10.1.1 Besondere Regeln für PCP 156
10.1.2 Besondere Regeln für MFT und MVT 156
10.2 Nachträgliche Definition von Dateien: DDNAME 157
10.3 Dateien im Ausgabestrom: SYSOUT. 160
10.3.1 DCB-Unterparameter 161
10.3.2 Besondere Regeln für PCP 162
10.3.3 Besondere Regeln für MFT und MVT 162
10.3.3.1 Die Parameter UNIT und SPACE 162
10.3.3.2 Besondere Verarbeitungsarten 163
10.4 Konventionelle E/A-Geräte 164
10.4.1 DCB-Unterparameter 164
10.4.2 Universaltypensatz 165
10.5 Graphische Geräte. 167
10.6 Pseudodateien: DUMMY, NULLFILE 167
10.7 Zusammenfassung 168

11. Speicher mit direktem Zugriff 170

11.1 Platzzuordnung 172
11.1.1. Anforderung von Platz: SPACE 172
11.1.1.1 Blockweise Anforderung von Platz: blocklänge 173
11.1.1.2 Spuren- oder zylinderweise Anforderung von Platz: TRK, CYL 175
11.1.1.3 Sekundär-Zuordnung: zuschlag 176
11.1.1.4 Platz für das Inhaltsverzeichnis oder die Indextabelle: directory, index. 178
11.1.1.5 Besondere SPACE-Optionen: RLSE, CONTIG, MXIG, ALX, ROUND 179
11.1.1.6 Platzfreigabe: RLSE. 179

11.1.1.7 Minimierung der Zugriffszeit: CONTIG, MXIG, ALX . . 180
11.1.1.8 Aufrunden auf ganze Zylinder: ROUND 181
11.1.2 Anforderung bestimmter Spuren: ABSTR 182
11.1.3 Zylinderaufteilung zwischen mehreren Dateien: SPLIT . 182
11.1.4 Weitervergabe von Platz: SUBALLOC 184
11.2 Datei-Kennsätze: LABEL 185
11.3 Dateien auf Speichern mit direktem Zugriff 186
11.4 Zugeordnete Arbeitsdateien 188
11.5 Zusammenfassung 189

12. Magnetbänder 191

12.1 Band-Kennsätze: LABEL 193
12.1.1 Dateifolgenummer 193
12.1.2 Art des Kennsatzes: typ 194
12.1.2.1 Standard-Kennsätze: SL 195
12.1.2.2. Nichtstandard-Kennsätze: NSL 196
12.1.2.3 Standard- und Benutzer-Kennsätze: SUL 196
12.1.2.4 Kennsatzverarbeitung überspringen: BLP 196
12.1.2.5 Keine Kennsätze: NL 197
12.1.3 Verarbeitung als Eingabe oder Ausgabe: IN/OUT 198
12.2 DCB-Unterparameter 199
12.2.1 Bandschreibdichte: DEN. 199
12.2.2 Bandaufzeichnungs-Technik: TRTCH 200
12.3 Verwendung von Bändern 201
12.4 Zusammenfassung 202

13. Der Linkage Editor 204

13.1 Der Linkage Editor 204
13.1.1 JCL für den Linkage Editor 206
13.1.2 Linkage Editor-Kontrollkarten 209
13.1.3 Erzeugung von Programmbibliotheken 212
13.1.4 Erzeugung von Unterprogrammbibliotheken 214
13.1.5 Überlagerungs-Strukturen. 217
13.1.5.1 Segmentierung des Programms 218
13.1.5.2 Die OVERLAY-Steuerkarte 219
13.1.5.3 Die INSERT-Steuerkarte 221
13.2. Verdichtung von Dateien 223
13.3 Zusammenfassung 226

14.	**Verschiedene JCL-Einrichtungen**	228
14.1	Befehls-Anweisung	228
14.2.	Null-Anweisung	229
14.3	Kommentar-Anweisung	229
14.4	Pflege der Dateien	230
14.4.1	Katalogisierte Dateien	230
14.4.2	Das Dienstprogramm IEHPROGM	231
14.4.2.1	Beispiele	237
14.5	Generation Data Groups	238
14.6	Dateien über mehrere Datenträger	241
14.7	Dateischutz: RETPD, EXPDT, PASSWORD	242
14.7.1	Schutz durch Aufbewahrungszeitraum: RETPD, EXPDT	243
14.7.2	Dateischutz durch Kennwort: PASSWORD	243
14.8	Steigerung der E/A-Leistungsfähigkeit	244
14.8.1	Schreibprüfung	244
14.8.2	Folgespureinrichtung	245
14.8.3	Sätze vom Typ Spanned Records	245
14.8.4	Standard-Blöcke	246
14.8.5	Verkettete Kanalprogramme (Chained Scheduling)	246
14.8.6	Kanaltrennung: SEP, AFF	247
14.9	Direkte Dateien	248
14.10	Index-sequentielle Dateien	250
14.10.1	Sätze	250
14.10.2	Blöcke	251
14.10.3	Spuren	251
14.10.4	Überlaufbereiche	252
14.10.5	Zylinderindex	253
14.10.6	Hauptindex	253
14.10.7	Zugriff zu index-sequentiellen Dateien	253
14.10.8	Erzeugung index-sequentieller Dateien	254
14.10.8.1	Der Parameter SPACE	255
14.10.8.2	Beispiel einer Platzzuordnung	256
14.10.8.3	Hauptdatenbereich	257
14.10.8.4	Indexbereich	259
14.10.8.5	Unabhängiger Überlaufbereich	261
14.10.9	Verwendung index-sequentieller Dateien	262
14.10.9.1	Indexbereich auf einem anderen Einheitentyp	265
14.10.9.2	Reorganisation index-sequentieller Dateien	266
14.11	Zusammenfassung	267

Anhang A: DCB-Unterparameter 268

Anhang B: Steuerzeichen 273

Anhang C: Zusammenfassung der JOB Control-Sprache 275

1. JOB-Karte 275
2. EXEC-Karte 277
3. Besondere DD-Karten 279
4. DD-Karte 280
5. Delimiter-Karte 285
6. PROC-Karte 285
7. Befehls-Karte 286
8. Null-Karte 286
9. Kommentar-Karte 286

Literaturhinweis 287

Sachwortverzeichnis 289

Vorwort

Dieses Buch ist für diejenigen bestimmt, die die Job Control-Sprache erlernen, verstehen und benutzen möchten. Es wendet sich an alle System/360-Programmierer, egal ob sie in COBOL, FORTRAN, PL/I, in Assembler-Sprache, RPG oder irgendeiner anderen Sprache kodieren.

Das Buch unterstellt bzw. es erwartet, daß der Leser ein aktiver System/360-Programmierer ist, der einige Vertrautheit im Umgang mit Rechenanlagen und Programmiersprachen besitzt. Dagegen wird vom Leser nicht erwartet, daß er mit dem System/360 vertraut ist.

Die Job Control-Sprache ist nicht leicht zu erlernen und nicht immer leicht anzuwenden. Dieses Buch eignet sich ebenso als Lehrtext zu ihrem Erlernen wie als Nachschlagewerk für diejenigen, die bereits mit ihr vertraut sind. Es kann zur Unterstützung eines Kurses dienen, der eine System/360-Rechenanlage verwendet.

Die Job Control-Sprache wird in dem großen Zusammenhang beschrieben, der zwischen ihr, dem Job des Benutzers und dem Betriebssystem besteht. Jede Eigenschaft der Job Control-Sprache wird vollständig detailliert erklärt, Beispiele für ihre Benutzung werden gegeben und mögliche Anwendungen werden diskutiert. Genauso werden viele Einrichtungen des System/360 ausführlich beschrieben und reichlich mit Beispielen versehen, um zu zeigen, wie sie mit der Job Control-Sprache eingesetzt werden können. Diese Einrichtungen schließen den Linkage Editor, index-sequentielle Dateien und mehrere, von der IBM zur Verfügung gestellte, Dienstprogramme ein.

Santa Monica, Kalifornien
Januar 1970

Gary DeWard Brown

Die Parameter der Job Control-Sprache

Parameter	Unterparameter von	Positions-/ Schlüsselwort-parameter	Seite
ABSTR	SPACE	P	
ACCT	EXEC	SCH	
AFF	DD	SCH	
AFF	UNIT	SCH	
BLKSIZE	DCB	SCH	
BUFNO	DCB	SCH	
CLASS	JOB	SCH	
CODE	DCB	SCH	
COND	JOB, EXEC	SCH	
CYLOFL	DCB	SCH	
DATA	DD	P	
DCB	DD	SCH	
DD	--	P	
ddname	DD	P	
DDNAME	DD	SCH	
DEN	DCB	SCH	
DISP	DD	SCH	
DPRTY	EXEC	SCH	
DSN oder DSNAME	DD	SCH	
DSORG	DCB	SCH	
DUMMY	DD	P	
EXEC	--	P	
EXPDT	LABEL	SCH	
JOB	--	P	
jobname	JOB	P	
JOBLIB	DD	P	

Parameter	Unterparameter von	Positions-/ Schlüsselwort-parameter	Seite
KEYLEN	DCB	SCH	
Kostenstellennummer	JOB	P	
LABEL	DD	SCH	
LRECL	DCB	SCH	
MODE	DCB	SCH	
MSGCLASS	JOB	SCH	
MSGLEVEL	JOB	SCH	
name	JOB	P	
NTM	DCB	SCH	
NULLFILE	DSN	P	
OPTCD	DCB	SCH	
PARM	EXEC	SCH	
PGM	EXEC	P	
PROC	--	SCH	
prozedur, PROC	EXEC	P	
PRTSP	DCB	SCH	
PRTY	JOB	SCH	
RD	JOB, EXEC	SCH	
RECFM	DCB	SCH	
REF	VOL	SCH	
REGION	JOB, EXEC	SCH	
RESTART	JOB	SCH	
RETPD	LABEL	SCH	
RKP	DCB	SCH	
ROLL	JOB, EXEC	SCH	
rückbezug	PGM, DCB, DSN, REF, SUBALLOC	P	
SEP	DD	SCH	
SEP	UNIT	SCH	
SER	VOL	SCH	
SPACE	DD	SCH	
SPLIT	DD	SCH	
STACK	DCB	SCH	
STEPLIB	DD	P	
stepname	EXEC	P	
SUBALLOC	DD	SCH	
SYSABEND	DD	P	

Parameter	Unterparameter von	Positions-/ Schlüsselwort parameter	Seite
SYSCHK	DD	P	
SYSOUT	DD	SCH	
SYSUDUMP	DD	P	
TIME	JOB, EXEC	SCH	
TRTCH	DCB	SCH	
TYPRUN	JOB	SCH	
UCS	DD	SCH	
UNIT	DD	SCH	
VOL oder VOLUME	DD	SCH	
*	DD	P	

1. Einführung

Das System/360 wurde 1964 als das Betriebssystem der Serie 360 der IBM-Rechenanlagen eingeführt. Bis heute hat sich die Aufmerksamkeit auf charakteristische Merkmale des Systems, wie Multiprogramming und die neue Programmiersprache PL/I, konzentriert. Wenig Beachtung dagegen fand die andere neue Sprache, die Job Control-Sprache. Trotzdem ist die JCL (Job Control Language) in vieler Hinsicht die bedeutendere Sprache, denn allein durch sie kann man die vielen Möglichkeiten des System/360 ausnutzen.

1.1 Die Rolle der JCL im System/360

JCL wird nicht gebraucht um Programme zu schreiben. Ganz im Gegenteil, sie besteht aus einem Steuerkartensatz, der dazu dient, Programme auszuführen und benutzte Ein-/Ausgabe-Geräte zu beschreiben. Traditionsgemäß waren Steuerkarten etwas, das vor oder hinter das Kartenpaket gelegt werden mußte, wollte man ein lauffähiges Programm erhalten. Jetzt aber werden diese Steuerkarten zur Sprache, der Job Control-Sprache; man muß sie programmieren, in einem Blockdiagramm festhalten, sie dokumentieren und austesten. Der Entwurf und das Kodieren von JCL-Anweisungen kann einen beträchtlichen Teil der Zeit des Programmierers beanspruchen.

1.2 Ziel und Zweck dieses Buches

Dieses Buch wurde für die Programmierer geschrieben, die das System/360 gerade erlernen und mit einer höheren Programmiersprache eini-

germaßen vertraut sind, die aber wenig JCL kennen. Es wurde genauso für die erfahrenen System/360-Programmierer geschrieben, die nichtsdestoweniger die volle Beherrschung der JCL noch nicht erreicht haben. Schließlich dient es dem in JCL versierten und erfahrenen Programmierer als Nachschlagewerk.

In diesem Buch werden einzelne JCL-Anweisungen sowie die Systemeinrichtungen, die von ihnen angesprochen werden, detailliert beschrieben. Da die JCL schwierig ist, wurden viele Beispiele eingefügt, die dem Leser beim Erlernen und Anwenden der JCL helfen.

Dieses Buch entspricht dem Release 18 des System/360 vom Oktober 1969. Obwohl IBM zweimal im Jahr neue Versionen des System/360 herausbringt, ändern die neuen Releases bestehende JCL gewöhnlich nicht. Jedoch werden manchmal kleinere Ergänzungen gemacht, so daß der Leser jedes neue Release daraufhin überprüfen sollte.

Viele der in diesem Buch beschriebenen Einrichtungen der JCL sind nur für bestimmte wählbare Systemzusätze sowie für Hardware Einrichtungen gültig. Im Text wird darauf besonders hingewiesen, und der Leser sollte sich nicht scheuen, die Seiten zu überschlagen, die für seine Bedürfnisse unerheblich sind.

Einführung in die JCL und das System/360

ient der Verständigung zwischen dem Benutzerprogramm und
m. JCL-Anweisungen sagen dem System, wie ein Job auszu-
sie fordern benötigte Systemeinrichtungen an und beschrei-
orderlichen Ein-/Ausgabe-Geräte (E/A-Geräte).

JCL-Karten

gsten JCL-Anweisungen (oder Karten) [1]) sind:

OB-Anweisung. Sie ist die erste Steuerkarte und kennzeich-
m Jobanfang.
XEC-Anweisung (Execute). Sie folgt der JOB-Karte und be-
die auszuführenden Programme oder Prozeduren.
D-Anweisung (Data Definition). Sie beschreibt jede Datei
ob Band-, Platten- oder Kartendatei) und verlangt vom System
ordnung (engl. Allocation) von E/A-Geräten.
elimiter-Anweisung (/*). Sie ist die End-of-File - Karte,
e für Dateiende), die das Ende einer Kartendatei anzeigt.

er benutzten JCL-Anweisungen lauten:

ull-Anweisung (//). Sie dient dazu, das Ende eines Jobs an-
gen. Ihre Verwendung ist aber nicht zwingend.
ommentar-Anweisung (//*). Kommentare können als Doku-
tionshilfe in den Spalten 4-80 abgelocht werden.
ROC-Anweisung. Diese Karte ordnet den symbolischen Para-
n in katalogisierten Verfahren (katalogisierten Prozeduren)

L-Anweisungen auch auf Band oder Platte gespeichert werden können, werden
eist in Karten abgelocht.

Anfangswerte zu (engl. Default, das sind die Werte, die angenommen werden, falls nichts anderes verfügt wird).

8) Die Befehls-Anweisung (Command). Diese Karte wird vom Operator benutzt, wenn er Operator-Befehle mit der JCL im Eingabestrom (Input Stream) eingeben möchte.

2.2 System/360: Aufbau und Begriffe

Das System/360 setzt sich aus neuen Rechenanlagen (der IBM 360), einem neuen Betriebssystem (dem OS/360) und einer neuen Terminologie zusammen. Da die Anwendung der JCL-Anweisungen das Verständnis von Aufbau und Begriffen des System/360 verlangt, ist der Rest dieses Kapitels ihrer Erklärung gewidmet.

2.3 Betriebssystem OS (Operating System)

Das *Betriebssystem OS* (*OS/360*) führt der Rechenanlage die Programme zu, leitet ihre Ausführung ein und stellt alle die Hilfsmittel und Dienstleistungen zur Verfügung, die die Programme verlangen. Da das Betriebssystem allgemein genug ausgelegt sein muß, um einer Vielfalt von Anwendungen, gefahren auf einer großen Reihe von Hardware-Konfigurationen, zu genügen, ist es - nämlich das Betriebssystem - aus einer allgemeinen Programmbibliothek aufgebaut, die auf viele Anforderungen zugeschnitten werden kann. Der Benutzer kann die von ihm benötigten Teile auswählen, ihnen seine eigenen Verfahren hinzufügen sowie diese Verfahren ändern, wenn sich ihre Voraussetzungen ändern sollten.

Die Programme und Routinen, die das Betriebssystem ausmachen, werden in Steuerprogramme und Verarbeitungsprogramme unterteilt. Die drei Hauptfunktionen der *Steuerprogramme* sind:

- Die Annahme und Planung der Jobs in ununterbrochener Folge (*Job Management*),
- die Überwachung jeder durchzuführenden Arbeitseinheit (*Task Management*) und
- die vereinfachte Wiederauffindung aller Daten, unabhängig von ihrer Organisation und Speicherung (*Data Management*).

Die *Verarbeitungsprogramme* bestehen aus:
- den *Sprachübersetzern* (wie z.B. den FORTRAN- und PL/I-Compilern),
- den *Dienstprogrammen* (wie z.B. dem Linkage Editor) und
- den *Anwendungsprogrammen* des Benutzers.

2.3.1 OS/360-Systemarten

Von der IBM werden drei Systemarten des OS/360 zur Verfügung gestellt:

- *PCP* Primary Control Program[2])
- *MFT* Multiprogramming with a Fixed Number of Tasks[2]) und
- *MVT* Multiprogramming with a Variable Number of Tasks[2]) .

Das PCP-System führt zu einer Zeit nur einen Job aus; unter MFT und MVT dagegen können mehrere Jobs konkurrierend ablaufen. PCP ist im allgemeinen für kleinere Rechner gedacht, MFT für mittlere und MVT für große Rechner.

2.3.2 Multiprogramming

Beide Systemarten MFT und MVT arbeiten mit Multiprogramming. Die Begriffe Multiprogramming, Multiprocessing und Time-Sharing werden oft durcheinandergeworfen. Beim *Multiprocessing* arbeiten zwei oder mehrere Zentraleinheiten am gleichen Speicher. *Time-Sharing* erlaubt die Benutzung einer Rechenanlage durch viele Benutzer zur gleichen Zeit, ohne daß einer von ihnen merkt, daß die Anlage gleichzeitig auch von anderen benutzt wird. Der Normalfall ist ein System mit mehreren direkt angeschlossenen Konsolen, die den einen Hauptrechner gleichzeitig benutzen. Time-Sharing versucht den Nutzen des Rechners für den einzelnen zu maximieren, nicht so sehr die Leistung des Rechners selbst.

Multiprogramming ist vom Konzept her gerade das Gegenteil. Es versucht die Leistung des Rechners zu maximieren, indem es seine wichtigsten Komponenten wie Zentraleinheit, E/A-Geräte und Speicher aus-

[2]) Anmerkung des Übersetzers
Die deutschen Termini lauten:
– Primäres Steuerprogramm
– Multiprogramming mit einer festen Anzahl von Tasks
– Multiprogramming mit einer variablen Anzahl von Tasks

lastet. Die meisten Jobs, die auf einer großen Anlage laufen, benutzen weder alle E/A-Geräte noch den ganzen Speicher. Ja noch mehr, nicht einmal die ganze Zentraleinheit wird benutzt, denn viel von der Laufzeit eines Jobs wird damit verbracht auf die Beendigung von E/A-Operationen zu warten. Das Rückspulen eines Magnetbandes ist dafür ein sehr treffendes Beispiel.

Da die meisten Jobs weder den ganzen Speicher noch alle E/A-Geräte benutzen, können sich beim Multiprogramming-Betrieb gleichzeitig mehrere Jobs im Speicher befinden, wobei zwischen ihnen hin und her geschaltet wird.

Mehrere Jobs werden also in den Speicher geladen; das Betriebssystem übergibt einem die Kontrolle, um sie, sobald dieser eine warten muß, einem anderen Job zu übergeben.

Das System kann mit der Einrichtung *Time-Slicing* ausgestattet werden; sobald ein angegebenes Zeitintervall verstrichen ist, wird zu einem anderen Job hinübergeschaltet, um zu verhindern, daß ein einzelner Job den Rechner blockiert.

Multiprogramming erlaubt E/A-intensiven Jobs, wie z.B. Sortierläufen, auf großen Rechenanlagen wirtschaftlich zu laufen. Diese Jobs belegen im allgemeinen wenig Speicher und verwenden den größten Teil ihrer Zeit für E/A-Operationen.

Viele der Einrichtungen einer Rechenanlage können gleichzeitig von mehreren im Speicher befindlichen, nicht aufeinander bezogener Jobs beansprucht werden. Andere Hilfsmittel dagegen müssen einem einzelnen Job zugeordnet werden. Das Betriebssystem muß seine Hilfsmittel so sorgfältig verteilen, daß die im Speicher befindlichen Jobs sich nicht um unteilbare Hilfsmittel, wie z.B. Bandeinheiten streiten. Es ist besser, einen Job zurückzuhalten, falls er ein nicht verfügbares Hilfsmittel beansprucht, und statt seiner einen anderen Job einzuplanen, dessen Hilfsmittelanforderung befriedigt werden kann.

Das System muß, wenn es die Jobs ihren Anforderungen entsprechend planen soll, wissen, welche Hilfsmittel jeder Job benötigt. Die Jobs geben dem System ihre Bedürfnisse durch die Verwendung von JCL-Anweisungen bekannt. Wenn ein Job beispielsweise ein Band braucht, wird eine JCL-Anweisung den benötigten Gerätetyp beschreiben, und das System wird den Job nicht laufen lassen, ehe nicht eine solche Bandeinheit verfügbar ist. Dadurch wird verhindert, daß der Job untätig im Speicher auf eine nicht verfügbare Bandeinheit wartet.

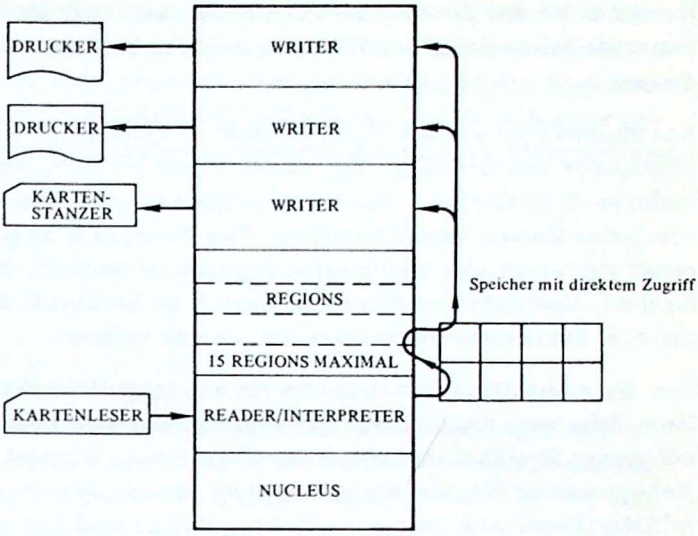


Abb. 2.1 Ein MULTIPROGRAMMING-SYSTEM

Abbildung 2.1 veranschaulicht ein Multiprogramming-System der 360. Jeder Job belegt einen zusammenhängenden Speicherbereich, *Region* (im MVT) oder *Partition* (im MFT) genannt; die Jobs bleiben solange im Speicher bis sie abgeschlossen sind. Einige Regions [3]), wie die der Reader und Writer schließen nie ab, sie bleiben stets im Speicher. Jede Region ist gegen Eingriffe von einer anderen Region geschützt. Das System bestimmt welche Region für wie lange laufen soll. Es ist so sicher wie nur möglich ausgelegt, so daß das System auch dann nicht zusammenbricht, wenn ein einzelner Job ausfällt.

Die erste Region enthält den *Nucleus* oder den residenten Teil des Betriebssystems (das sind die Teile des Systems, die nicht auf Speichern mit direktem Zugriff - wie Platte oder Trommel - gehalten werden). Der *Reader/Interpreter* (kurz *Reader*) liest die Jobs vom Kartenleser ein und reiht sie auf einem Speicher mit direktem Zugriff in eine Warteschlange ein. Die *Writer* drucken die Ausgabe auf der richtigen Ausgabeeinheit aus, indem sie die Ausgabe von der Warteschlange auf dem Speicher mit direktem Zugriff abrufen. Das Einordnen von Eingaben und Ausgaben in Warteschlangen, die auf Speichern mit direktem Zugriff liegen, wird *Spooling* genannt. Die Folge der vom Reader eingelesenen

[3]) Der Ausdruck Region ist dem Ausdruck Partition in den folgenden Erläuterungen gleichwertig.

Karten heißt der *Eingabestrom* (*Input Stream*), und die Folge der Ausgaben, rausgeschrieben von den Writern, heißt der *Ausgabestrom* (*Output Stream*).

Normalerweise werden *konventionelle E/A-Geräte*, wie Kartenleser, Kartenstanzer und Drucker, den Readern und Writern zugeordnet. Alle anderen E/A-Geräte - von Bändern abgesehen - können konkurrierend von jeder Region benutzt werden. Das Bandgerät kann zu einem Zeitpunkt nur einer einzigen Region zugeordnet werden. Dies ist einer der Gründe, Speicher mit direktem Zugriff zu benutzen; der Job wird nicht auf eine freiwerdende Bandeinheit warten müssen.

Der Betriebsablauf des Systems ist wie folgt: Der Reader/Interpreter liest Jobs vom Kartenleser ein und ordnet sie in eine Warteschlange auf einem Speicher mit direktem Zugriff ein. Kommt ein Job an den Anfang seiner Warteschlange, so lädt ihn ein Systemprogramm - *Initiator/Terminator* genannt - in eine Region und führt ihn aus. Falls dieser Job Datenkarten liest, holt das System diese Karten von dem Speicher mit direktem Zugriff, auf dem sie zwischengespeichert wurden. (Karten von einem Speicher mit direktem Zugriff zu lesen geht schneller als sie unmittelbar vom Kartenleser zu lesen). Sollte der Job eine Liste drucken oder Karten stanzen, so wird diese Ausgabe wiederum auf einem Speicher mit direktem Zugriff zwischengespeichert. Ein Job wird nie unmittelbar Karten lesen oder stanzen, noch unmittelbar drukken; das geschieht stets durch Zwischenspeicherung in Warteschlangen, die auf Speichern mit direktem Zugriff liegen. Der Job wird dessen nicht gewahr, da das System dies für ihn automatisch erledigt. Nachdem die Ausgabe auf dem Speicher mit direktem Zugriff zwischengespeichert worden ist, gibt sie der Writer auf der richtigen Ausgabeeinheit aus.

Obwohl hier nur der Jobablauf in einer einzigen Region beschrieben wurde, können in entsprechender Weise gleichzeitig mehrere Regions laufen.

MFT und MVT unterscheiden sich grundsätzlich dadurch, daß MFT fest eingeteilte Partitions hat. Die Größe und die Anzahl der Partitions wird festgesetzt, wenn das MFT-System generiert wird. Der Operator kann sie jedoch vom Konsol aus ändern. Die Anzahl der Partitions (ihr Maximum ist 15) begrenzt die Zahl der konkurrierend ablaufenden Jobs; jeder einzelne Job läuft in einer Partition.

MVT ordnet jedem Jobstep automatisch eine Region zu, ausgehend von der vom Step verlangten Speichermenge. (Ein MFT-Job beschlagnahmt

allen Speicher in einer Partition, egal ob er ihn benutzt oder nicht.) Die Anzahl der konkurrierend laufenden Jobs (ihr Maximum ist 15) wird von der Zahl der Jobs begrenzt, die gleichzeitig in den Speicher passen. MVT besitzt die Einrichtung *Subtasking*, die es dem sehr gewandten Programmierer gestattet, innerhalb seines Jobs Multiprogramming zu betreiben.

2.4 Jobs

Der *Job* ist für die Rechenanlage die selbständige Grundeinheit der zu bewältigenden Arbeit. Er besteht aus einer Gruppe von Anweisungen, geschrieben in einer Programmiersprache (man spricht sehr oft von den *Anweisungen des Quellenprogramms*). Die Anweisungen werden auf speziellen Formblättern kodiert. Anschließend werden sie in Karten abgelocht und der Rechenanlage eingegeben. Man nennt diesen Satz von Anweisungen der Quellensprache, die zusammen im Rahmen eines Jobs ausgeführt werden, ein *Programm*. Programme werden gewöhnlich in Funktionseinheiten unterteilt, die *Module, Subroutinen* oder *Unterprogramme* genannt werden. Man tut dies, um die verschiedenen Teile austesten und gegebenenfalls ändern zu können, ohne daß die anderen Teile davon beeinflußt werden.

JCL-Karten weisen das Betriebssystem an, die Verarbeitung für einen Job durchzuführen, und sie beschreiben alle dazu benötigten E/A-Geräte. Da diese JCL-Anweisungen komplex sind, und da es viele sind, speichert man die Karten für häufig benutzte Prozeduren auf einem Speicher mit direktem Zugriff. Statt die Karten immer wieder direkt einzugeben, ruft der Benutzer nur noch diese sogenannten *katalogisierten Verfahren* auf, indem er dem System ihren Namen nennt.

Jeder Job muß mit einer einzelnen JOB-Karte beginnen.[4])

```
//TEST#7   JOB 6245,MUELLER
```

Diese Anweisung sagt dem Betriebssystem, daß ein Job namens TEST # 7 der Kostenstelle 6245 belastet werden soll, und daß er einem Programmierer namens Müller gehört.

---

[4]) Die JOB-Karte, wie auch alle anderen JCL-Karten, werden in den folgenden Kapiteln beschrieben.

2.4.1 Jobsteps

Jede Ausführung eines Programms innerhalb eines Jobs heißt ein *Jobstep* oder einfach ein *Step*; ein Job kann demnach aus mehreren Steps bestehen. Ein sehr typischer Job ist der Compile-Link-Go-Job. Er besteht aus dem Compile-Step, der die Anweisung der Quellensprache in Maschinensprache umwandelt, aus dem Linkage Editor-Step, der das umgewandelte Programm mit anderen Programmen aus Unterprogrammbibliotheken verbindet, und aus dem Go-Step, dem eigentlichen Ausführungsstep, in dem das Programm nun tatsächlich läuft. Für diesen Job bedeutet dies also die Ausführung dreier getrennter Programme: des Compilers und des Linkage Editors, die Systemprogramme sind, und des Benutzerprogramms. Alle drei aber benötigen JCL-Anweisungen.

Jeder Jobstep beginnt mit einer einzelnen EXEC-Karte, die ein Programm oder ein katalogisiertes Verfahren benennt.

```
//STEP1    EXEC PGM=PL1COMP
//STEP2    EXEC PL1CLG
```

STEP1 führt das Programm PL1COMP aus; STEP2 dagegen ruft das katalogisierte Verfahren PL1CLG auf.

Die Steps innerhalb eines Jobs werden sequentiell nacheinander ausgeführt, so daß die Ausgabe eines einzelnen Steps die Eingabe des folgenden werden kann. PCP- und MFT-Systeme führen jeden Jobstep eines Jobs im gleichen Kernspeicherbereich aus. MVT dagegen ordnet jedem Jobstep - basierend auf seiner Speicherplatzanforderung - eine neue Region zu.

Das folgende Beispiel veranschaulicht den Unterschied zwischen Jobs und Jobsteps.

```
//LAUFA    JOB 7233,MUELLER
//COMP     EXEC PGM=FORTCOMP
//LKED     EXEC PGM=LINKEDIT
//VERSUCH1 JOB 6221,MAIER
//PL1      EXEC PGM=PL1COMP
//LKED     EXEC PGM=LINKEDIT
```

Es werden zwei verschiedene Jobs, nämlich LAUFA und VERSUCH1, ausgeführt. Die Steps innerhalb eines Jobs werden nacheinander ausgeführt. Die Jobs jedoch brauchen in den Multiprogramming-Systemen nicht nacheinander abzulaufen. VERSUCH1 könnte vor LAUFA oder nach ihm ausgeführt werden. Schließlich könnten beide Jobs konkurrierend miteinander ablaufen.

2.4.2 Compilation

Ein Kartenpaket, das Anweisungen einer Quellensprache enthält, wird ein *Quellenpaket* oder ein *Quellenmodul* genannt. Der *Compiler* - ein System/360-Programm liest das Quellenmodul ein und übersetzt die Sprachanweisungen in Maschineninstruktionen. Die Compiler-Ausgabe, die aus den übersetzten Maschinenbefehlen besteht, heißt *Objektmodul.* Gewöhnlich speichert der Compiler das Objektmodul auf einem Speicher mit direktem Zugriff für die nachfolgende Verarbeitung desselben. Der Benutzer kann den Compiler aber auch anweisen, das Objektmodul in Karten abzustanzen um es so aufzubewahren. Ein solches Kartenpaket heißt *Objektpaket.*

2.4.3 Linkage Editor

Bevor das Objektmodul ausgeführt werden kann, muß es vom *Linkage Editor* verarbeitet werden. Der Linkage Editor verarbeitet Objektmodule, die ihm von einem Compiler übergeben werden, aber ebenso bereits übersetzte Objektmodule, die vom Kartenleser eingelesen werden. Der Linkage Editor ist ein Systemprogramm, das feststellt, welche Programmteile andere Unterprogramme aufrufen, und das diese sogenannten *externen Referenzen* auflöst. (Unter externer Referenz versteht man Symbole in einem Programmteil, auf die von einem anderen Programmteil Bezug genommen wird.) Der Linkage Editor durchsucht ebenso die verschiedensten *Unterprogrammbibliotheken,* um die Unterprogramme zu beschaffen, die zusätzlich nötig sind einen Job zu vervollständigen. (Unterprogrammbibliotheken sind Dateien ausgeprüfter Unterprogramme, gespeichert auf Speichern mit direktem Zugriff.)

Bibliotheken, die allen zugänglich sind, wie z.B. die FORTRAN- oder PL/1-Bibliothek, werden gewöhnlich automatisch durchsucht. Der Benutzer kann sich daneben eine *private Bibliothek* anlegen, die seine ei-

genen Unterprogramme enthält. Er kann den Linkage Editor anweisen, auch diese zu durchsuchen.

Der Linkage Editor fügt diese Unterprogramme zu einem vollständigen Programm, dem *Lademodul* zusammen. Das Lademodul kann sofort vom Speicher mit direktem Zugriff in den Kernspeicher geladen und ausgeführt werden. Viele Programme, die zu groß sind, um in den Kernspeicher zu passen, werden in *Segmente* unterteilt, die nicht alle gleichzeitig im Speicher zu sein brauchen. Der Benutzer weist den Linkage Editor durch besondere Steuerkarten an, das Programm in Segmente zu unterteilen, die einander *überlagern* können, so daß der Speicher immer wieder von verschiedenen Segmenten benutzt werden kann. Eine so *geplante Überlagerungsstruktur* kann den Kernspeicher sehr wirksam nutzen. Sie ist bei FORTRAN-, PL/I- und Assembler-Programmen möglich, nicht dagegen bei ALGOL- und RPG-Programmen.

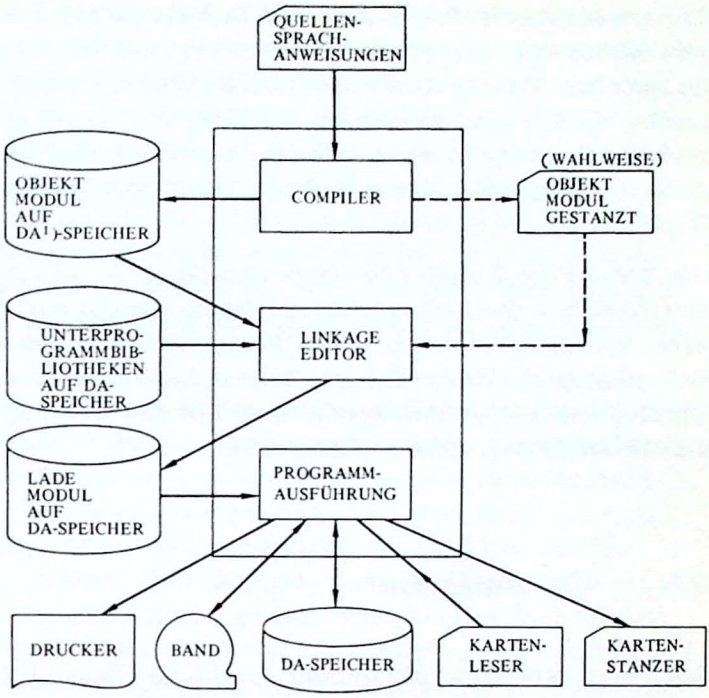
Andere ebenso große Programme dagegen lassen sich nicht so leicht segmentieren, weil sich die Reihenfolge, in der die verschiedenen Programmmodule ausgeführt werden, von vorneherein nicht bestimmen läßt. Hier gestattet es die *dynamische Struktur*, die Lademodule während des Programmablaufes in den Kernspeicher zu bringen, und zwar dann, wenn sie benötigt werden. Die dynamische Struktur kann in allen Sprachen, außer FORTRAN und RPG, benutzt werden.

2.4.4 Ausführung

Nachdem das Programm vom Linkage Editor verarbeitet worden ist, wird es in den Kernspeicher geladen und ausgeführt.

Dieser Step wird gewöhnlich der *Go-Step* genannt. Wenn das Programm ausgeprüft worden ist, kann der Benutzer es als Lademodul auf einem Speicher mit direktem Zugriff abspeichern; er spart auf diese Weise Übersetzungs- und Linkage Editor - Verarbeitungszeit. Um die Ausführung eines Lademoduls einzuleiten, ist nur der Bruchteil einer Sekunde erforderlich, während es viele Sekunden dauern kann, das gleiche Programm zu übersetzen und mit dem Linkage Editor zu verarbeiten. Abbildung 2.2 veranschaulicht die drei Steps: Compile-, Linkage Editor- und Go-Step.

Ein Job braucht nicht aus dem Compile-, Linkage Editor- und Go-Step zu bestehen. Es genügt vielleicht der Übersetzungsstep für die Quellenanweisungen, um hier Formalfehler auszuprüfen. Vielleicht sind aber auch mehrere aufeinander bezogene Ausführungssteps erforderlich, wo-



[1]) DA-Speicher ist Direkt-Access Storage d.h. Speicher mit direktem Zugriff

Abb. 2.2 SYSTEM/360 JOB Ausführung

bei jeder Step nur einige Daten verarbeitet und die Ergebnisse an einen der nachfolgenden Steps weiterleitet. Z.B. könnte der erste Step eines automatischen Adressierungssystems die Namen und Adressen in einem Stammbestand fortschreiben. Der zweite Step könnte die Namen und Adressen eines bestimmten Ziehkodes aussteuern. Der letzte Step endlich würde die ausgewählten Namen und Adressen auf Briefaufkleber drucken können.

2.4.5 Der Loader

Ein besonderes Programm - der *Loader* - verbindet die Verarbeitung des Linkage Editor mit der Ausführung in einem Step. Wie der Linkage Editor nimmt der Loader Objektmodule an, die ihm vom Übersetzer übergeben werden, ebenso aber auch vorher übersetzte Objektmodule in Kartenform über den Leser. Er löst ebenfalls externe Referenzen und durchsucht Unterprogrammbibliotheken.

Der Loader unterscheidet sich vom Linkage Editor dadurch, daß er kein Lademodul erzeugt. Statt dessen verarbeitet er alle Objektmodule im Speicher. Der Benutzer muß ausreichend Kernspeicher für den Loader und für sein Programm bereitstellen. Wenn die Verarbeitung beendet ist, übergibt der Loader die Kontrolle dem Programm und die Ausführung beginnt. Diesen Vorgang nennt man *Laden und Ausführen* (*Load and Go*).

Der Vorteil des Loaders ist seine Schnelligkeit, er ist beinahe zweimal so schnell wie der Linkage Editor. Da im ganzen nur ein Step benötigt wird, verringert sich der Verwaltungsanteil, d.h. es wird noch einmal Zeit eingespart. Nachteilig ist, daß ein Lademodul für die spätere Ausführung nicht aufbewahrt werden kann und daß Überlagerungs- und dynamische Strukturen nicht zulässig sind.

2.5 Ein-/Ausgabe

Ein Programm wird während der Ausführung Daten vom Kartenleser, vom Band oder vom Speicher mit direktem Zugriff einlesen und es wird Daten ausgeben auf den Drucker, auf den Kartenstanzer, auf Band oder auf einen Speicher mit direktem Zugriff. Das Programm wird zur Ausführung vielleicht auch *Hilfsspeicher* benötigen, der vorübergehend Daten aufnehmen soll.

2.5.1 Datenorganisation

Jede benannte Menge von Daten (ein Kartenpaket, ein Quellen- oder Objektmodul, oder schlechthin Daten) heißt *Datei*. Die Begriffe Datei und Datenbestand haben die gleiche Bedeutung. Dateien können sequentiell, untergliedert, direkt oder index-sequentiell organisiert sein. Um die Unterschiede zu verstehen vergegenwärtige man sich verschiedene Kartenpakete. Die *sequentielle* Organisation besteht darin, die Kartenpakete übereinander zu legen und die Karten, eine nach der anderen, in der Reihenfolge zu verarbeiten, in der sie im Gesamtpaket liegen. Das Magnetband, der Drucker, der Kartenleser und der Speicher mit direktem Zugriff können sequentielle Dateien aufnehmen. Die sequentielle Organisation ist dann angebracht, wenn die Datei in der Reihenfolge verarbeitet werden muß, in der sie gespeichert ist.

Wenn die Kartenpakete in getrennten Ablagefächern liegen, kann man ein bestimmtes Kartenpaket aufnehmen und verarbeiten, indem man es sequentiell liest. Diese *untergliederte* Organisation, bei der man die einzelnen Kartenpakete *Members* nennt, wird am häufigsten für Unterprogrammbibliotheken verwendet, da ein Unterprogramm (ein Member dieser Bibliothek) angesprochen oder auch ersetzt werden kann, ohne daß die anderen verändert werden. Eine untergliederte Datei kann sequentiell verarbeitet werden, indem man die Members hintereinander eins an das andere hängt (*concatenating* the members). Untergliederte Dateien können nur auf Speichern mit direktem Zugriff gespeichert werden.

Wenn alle Karten eines Kartenpaketes auf einem Tisch so ausgebreitet werden, daß jede einzelne Karte zu sehen ist, so kann man jede Karte direkt ansprechen und ersetzen, ohne daß die anderen davon berührt werden. Die *direkte* Dateiorganisation ist nur auf Speichern mit direktem Zugriff möglich, sie ist dort angebracht, wo die Sätze *wahlfrei* verarbeitet werden müssen. Das bedeutet, daß der nächste zu verarbeitende Satz keinerlei Beziehung zum gerade verarbeiteten Satz besitzt. Einige Sprachen, wie COBOL, PL/I und FORTRAN, haben Anweisungen für die direkt organisierte Datei, die diese komplizierte Organisation für den Anwender sehr vereinfachen.

Ein gutes Analogon für *index-sequentielle* Dateien ist die öffentliche Bibliothek, in der jedem Buch ein *Schlüssel* (eine Zahl aus der Dewey'schen Dezimalklassifikation für Bücher) zugeordnet wird, und in der die Bücher in der *Sortierfolge*, basierend auf dem Schlüssel, eingeordnet sind. Die Bücher stehen in Regalen, sie können weggenommen oder eingeordnet werden, ohne daß die Bücher auf den anderen Regalen davon berührt werden. Man kann diese "Datei" sequentiell verarbeiten, indem man durch die Regale schlendert, man kann aber ein bestimmtes Buch auch direkt herausgreifen, indem man es über den *Index* sucht. Indexsequentielle Dateien können mit Hilfe der Assembler-Sprache, mit PL/I, COBOL und RPG erzeugt werden.

2.5.2 Daten im Rechner

Die Daten im Speicher der Rechenanlage bestehen aus Bits, Bytes, Worten, Sätzen und Blöcken. Ein *Bit* ist eine einzelne Binärziffer, nämlich 0 oder 1. Ein *Byte* ist die kleinste adressierbare Einheit im System/360, es besteht aus 8 Bits. Alle Kernspeichergrößen und Speicherkapa-

zitäten im System/360 werden in Bytes ausgedrückt. Ein *Wort* enthält 32 Bits bzw. 4 Bytes, den zur Speicherung einer Zahl von einfacher Genauigkeit benötigten Kernspeicher. Das Wort war die Standardeinheit des Kernspeichers der Rechner der zweiten Generation, weil es der für eine Maschineninstruktion benötigte Speicherplatz war. Es wurde durch das Byte der Maschinen der dritten Generation, wie z.B. des Systems/360, ersetzt, weil mehr als eine Instruktion in einem Wort gespeichert werden kann, und weil ganze Zahlen oder Kommazahlen verschiedene Wortgrößen einnehmen würden.

Da es sehr mühselig ist binäre Zahlen zu schreiben (ein Byte, das eine binäre 1 enthält, muß als 0000 0001 geschrieben werden), wird die *hexadezimale* Schreibweise verwendet (ihre Basis ist 16). Eine hexadezimale Ziffer besitzt 4 Bits, d.h. 2 hexadezimale Ziffern stellen ein Byte dar. Die binären Äquivalente der hexadezimalen Ziffern sind:

Binär	Hex	Binär	Hex
0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F

Z.B. würden wir die binäre Zahl 00110100011101001111000011011011 hexadezimal als 3474F0DB schreiben, was Platz spart und die Fehler beim Abschreiben der Zahl verringert. Um binäre Daten des System/360 darzustellen, wird im allgemeinen die hexadezimale Schreibweise verwendet.

Die System/360-Verschlüsselung eines Zeichens besteht aus 8 Bits, so daß ein Byte gerade ein Zeichen aufnimmt. Um die Zeichen intern im Rechner darzustellen, wird der *EBCDIC*-Kode verwendet (Extended Binary-Coded-Decimal Interchange Code). Die Zeichen unterteilen sich in Abhängigkeit vom Zusammenhang in verschiedene Zeichensätze. Die JCL benutzt die *alphamerischen* Zeichen A bis Z und 0 bis 9, die *nationalen* Zeichen @, $ und # sowie die *Sonderzeichen* , . / ') (* & + - = und Blank (Leerstelle).

Der *Satz* ist die logische Einheit der Daten, er kann aus einer beliebigen Anzahl von Bytes bestehen; beispielsweise enthält der Lohn- oder Gehaltssatz alle auf den Empfänger bezogenen Daten, wie den Namen, das Gehalt und die Freibeträge. Die Begriffe Satz und *logischer Satz* werden für den gleichen Sachverhalt benutzt.

Die Sätze des System/360 haben drei verschiedene Formate: Fest, variabel und nicht definiert. *Feste* Sätze sind alle von der gleichen Länge; beispielsweise hat eine Lochkartendatei Sätze fester Länge, weil jede Karte 80 Zeichen enthält.

Variable Sätze haben unterschiedliche Länge, wobei die ersten 4 Bytes des Satzes seine Länge angeben. Z.B. könnte eine Personaldatei, die die Namen der Mitarbeiter enthält, aus variabel langen Sätzen bestehen, da die Namen der Mitarbeiter ja unterschiedlich lang sind.

Die *nicht definierten* Sätze haben unterschiedliche Länge, ohne daß hier die Längenangabe im Satz enthalten ist. Die Sätze werden auf dem Speichermedium durch eine physische *Kluft* (*Gap*) voneinander getrennt. Der Rechner ist bei der Übertragung des Blockes in der Lage diese Kluft zu erkennen, und damit auch in der Lage, die Blöcke voneinander zu trennen. Sätze nicht definierter Länge werden verwendet, wenn die Satzlänge erst nach dem Schreiben des ganzen Satzes bekannt ist, wie etwa an der Konsolschreibmaschine.

Die Daten werden zwischen dem Speicher und den E/A-Geräten in Blökken übertragen. Jeder Block wird von anderen Blöcken durch eine Kluft getrennt. Der einzelne Block kann mehrere Sätze fester oder variabler Länge enthalten, dagegen ist der Satz nicht definierter Länge mit dem Block identisch. Das bedeutet, daß ein *Block* aus einem oder mehreren Sätzen, die auf einmal übertragen werden, besteht. Sätze fester Länge können etwas schneller verarbeitet werden als Sätze variabler Länge, da sie leichter zu blocken und entblocken sind. In einem Block mit Sätzen variabler Länge geben die ersten 4 Bytes die Länge des Blockes an.

Daten können sehr schnell zwischen dem Kernspeicher und einem Speicher mit direktem Zugriff oder einem Magnetband übertragen werden (bis zu 1.2 Millionen Bytes pro Sekunde), vorausgesetzt, die Übertragung der Daten hat tatsächlich begonnen. Dagegen dauert es wegen der mechanischen Trägheit vergleichsweise lange (mehr als 50 Millisekunden) die Übertragung einzuleiten. Man denke etwa an die Umdrehungszeit eines Plattenspeichers. Die Blockung gestattet es, große und damit wirksame Datengruppen auf einmal zu übertragen. Sie spart außerdem

auf dem Speichermedium Platz, weil sie die Anzahl der Klüfte herabsetzt.

Die Anzahl der Sätze pro Block heißt *Blockungsfaktor.* Der Block wird manchmal auch *physischer Satz* genannt, doch kann dieser Ausdruck leicht mit dem logischen Satz verwechselt werden. Da man nur zur besseren Ausnutzung der Hardware blockt, das Blocken aber in keinerlei Beziehung zur Art und Weise steht, in der der Benutzer seine Daten verarbeitet, so übernimmt gewöhnlich das Betriebssystem Blocken und Entblocken.

Der Block wird in einen Speicherbereich, *Puffer* genannt, eingelesen. Sobald der letzte Satz des Blockes verarbeitet worden ist, liest das Betriebssystem den nächsten Block ein. Das Entgegengesetzte geschieht beim Schreiben der Daten. Es ist möglich mehrere Puffer anzufordern, so daß das Betriebssystem den nächsten Puffer einlesen kann, während die Daten aus dem ersten Puffer verarbeitet werden. Daraus resultiert eine beträchtliche Steigerung der Verarbeitungsgeschwindigkeit, weil die Ein-/Ausgabe *überlappt* wird; d.h. Daten können gelesen oder geschrieben werden während gleichzeitig Rechnungen im Speicher durchgeführt werden.

Dateien müssen *eröffnet* werden, bevor man sie benutzen kann, sie müssen nach beendeter Verarbeitung *geschlossen* werden. Das Betriebssystem eröffnet Dateien automatisch beim ersten Versuch sie zu benutzen, es schließt sie, wenn der Job endet. Jedoch können Dateien im Assembler, in COBOL und in PL/I auch explizit geöffnet und geschlossen werden (OPEN, CLOSE).

Wenn die Datei *eröffnet* wird, erzeugt das Betriebssystem alle internen Kontrollblöcke, die nötig sind, um die Ein-/Ausgabe durchzuführen, es positioniert die Datei auf ihren Anfang und macht sie so zur Verarbeitung fertig. Das *Schließen* der Datei gibt alle Puffer und Kontrollblöcke, die der Datei zugeordnet waren, wieder frei, es stellt benutzte Bandeinheiten und Zwischenspeicherplatz auf Platten wieder zur Verfügung; allgemein gesprochen stellt es nach Verarbeitung der Datei den alten Zustand wieder her.

2.5.3 Zugriffstechniken

Unter *Zugriffstechnik* versteht man die Methode, in der die Daten zwischen Kernspeicher und E/A-Geräten bewegt werden. Im Betriebssystem OS werden zwei dieser Techniken, nämlich Basic und Queued, zur Verfü-

gung gestellt. Bei der *Basic-Zugriffstechnik* startet ein READ- bzw. ein WRITE-Befehl die Übertragung zwischen Kernspeicher und E/A-Gerät. Da die E/A-Operationen überlappt ablaufen, kann das Programm seine Ausführung fortsetzen, während die E/A-Operation abläuft. Es liegt in der Verantwortlichkeit des Benutzers Puffer zur Verfügung zu stellen, sowie die Ein-/Ausgabe mit dem Programm zu *synchronisieren* (d.h. sicherzustellen, daß ein Puffer vollständig rausgeschrieben wurde, bevor neue Daten in ihn hineingelangen). Die Basic-Zugriffstechnik gibt dem Benutzer die Möglichkeit der nahezu vollständigen Kontrolle der E/A, sie verlangt dafür aber auch mehr Programmieraufwand.

Bei der *Queued-Zugriffstechnik* sorgt das System vollständig für Pufferung und Synchronisation. Das Programm verlangt mit GET oder gibt mit PUT einen logischen Satz, und das Betriebssystem erledigt die ganze Ein-/Ausgabe, das Puffern, Blocken und Entblocken. Die Queued-Technik ist am leichtesten zu verwenden, sie wird in den meisten höheren Programmiersprachen zur Verfügung gestellt.

2.6 Datenspeicherung

Die drei am häufigsten benutzten Speicher sind die Lochkarte, das Magnetband und der Speicher mit direktem Zugriff.

2.6.1 Karten

Die einzelne Karte enthält 80 Zeichen. Die meisten Programme und ihre Daten werden anfangs auf Karten abgelocht. Karten sind leicht zu handhaben, leicht zu ändern und sie sind außerdem billig. Auf der anderen Seite sind große Kartenpakete unhandlich, schwer und wenig wirtschaftlich. Die Karten-Ein-/Ausgabe ist sequentiell, das bedeutet, will man die zehnte Karte lesen, muß man auch die neun vorausgehenden Karten lesen.

2.6.2 Magnetbänder

Die zur Speicherung von Rechnerdaten benutzten Magnetbänder sind den zuhause in Bandgeräten benutzten ähnlich, wenngleich die Art der Speicherung und der Inhalt verschieden sind. Ein Datenbyte wird quer über

die Bandbreite gespeichert; die Position jedes Bits quer über die Bandbreite nennt man eine *Spur*. Die *Speicherdichte* ist die Entfernung zwischen aufeinanderfolgenden Bits in der Längsrichtung des Bandes. Daten können auf dem Band mit einer Dichte von 200, 556, 800 oder 1600 bpi (bits per inch) aufgezeichnet werden, d.h. auf einem Magnetband von 1 inch Länge können auf jeder Spur 200, 556, 800 oder 1600 Bits gespeichert werden. Ältere IBM-Rechenanlagen besaßen 6-Bit Bytes, und die Daten wurden auf 7-Spur Magnetbändern gespeichert (ein 7. Bit wurde als Prüfbit hinzugefügt). Da die System/360-Rechenanlagen 8-Bit Bytes besitzen, werden die Daten nun auf 9-Spur Bändern aufgezeichnet.

Eine einzelne Magnetbandrolle kann bis zu 26 Millionen Bytes aufnehmen; benutzt man mehrere Rollen, so ist die speicherbare Informationsmenge unbegrenzt. Magnetbänder werden, genau wie Lochkarten, dazu benutzt, sequentielle Daten zu speichern. Obwohl mehrere Dateien auf einer einzelnen Bandrolle gespeichert werden können, indem man sie durch Bandmarken voneinander trennt, macht man das selten, weil sich Bänder schwer fortschreiben lassen und jede Änderung das Neuschreiben des ganzen Bandes erforderlich macht. Da die Bänder durch das Kopieren des alten Bandes zusammen mit den Änderungen auf ein neues Band fortgeschrieben werden, hat man mit dem Aufbewahren des alten Bandes automatisch die Daten gesichert.

Bänder sind ein ausgezeichneter Langzeitspeicher, denn die Bandrolle ist billig und sie kann auf kleinem Platz viel Informationen enthalten. Bänder lassen sich beträchtlich schneller als Karten verarbeiten, sie können schneller oder langsamer als Speicher mit direktem Zugriff sein, je nach benutzter Bandeinheit.

2.6.3 Speicher mit direktem Zugriff

Speicher mit direktem Zugriff sind die vielseitigsten Speicher, sie können sequentielle, untergliederte, direkte und index-sequentielle Daten enthalten. Speicher mit direktem Zugriff leiten ihren Namen von der Art und Weise ab, mit der zu den Daten zugegriffen wird. Im Gegensatz zu Lochkarten und Magnetbändern muß man die ersten neun Sätze nicht lesen um den zehnten Satz zu bekommen.

Über die Speicherung allgemeiner Daten hinaus wird der Speicher mit direktem Zugriff als Hilfsspeicher benutzt sowie zur Speicherung von Unterprogrammbibliotheken oder zur Speicherung vollständiger Pro-

gramme in Lademodulform. Auch nichtresidente Teile des Betriebssystems werden auf Speichern mit direktem Zugriff gespeichert.

Speicher mit direktem Zugriff sind Trommel-, Platten- und Streifenspeichereinheiten. *Trommeln* sind die schnellsten Speicher mit direktem Zugriff, können aber die wenigsten Daten enthalten. Eine Trommel kann auch nicht entladen werden, um eine andere Trommel an ihrer Stelle zu laden. Die *Platte* ist ein verhältnismäßig schneller Speicher, und sie kann eine große Menge von Daten aufnehmen. Die Platteneinheiten 2311 und 2314 besitzen entladbare Plattenstapel, die man leicht entfernen und durch andere ersetzen kann, wodurch die Datenmenge praktisch unbegrenzt ist. *Streifenspeichereinheiten* sind sehr langsam, sie können aber eine immense Datenmenge speichern. Der Streifenspeicher kann nur sequentielle Daten aufnehmen.

Der Plattenspeicher besteht, ähnlich einem Schallplattenstapel, aus einem Stapel rotierender Platten mit bespeicherbaren Oberflächen. Jede Plattenoberfläche enthält viele kreisförmige, konzentrisch angeordnete *Spuren*. Die Spuren sind den Rillen einer Schallplatte ähnlich, mit dem Unterschied, daß die Schallplattenrillen eine fortlaufende Spurrille bilden, die sie zu einer sequentiellen Einrichtung macht.

Ein Satz elektronischer Lese-Schreib-Köpfe befindet sich zwischen den Plattenoberflächen, und ist auf einem *Zugriffsarm* montiert. Wenn eine bestimmte Spur gelesen oder beschrieben werden soll, wird der Zugriffsarm so bewegt, daß der Lese-Schreib-Kopf über dieser Spur zu stehen kommt. Diese Armbewegung heißt eine *Suchbewegung* (*Seek*). Der Lese-Schreib-Kopf sucht nach einer besonderen Markierung auf der rotierenden Spur, die ihm sagt, wo die Spur beginnt. Dadurch gibt es zwei mechanische Verzögerungen beim Zugriff zu einer bestimmten Spur: Die Verzögerung bei der Suchbewegung, die davon abhängt, wie weit der Zugriffsarm bewegt werden muß und die Rotationsverzögerung, die im Durchschnitt eine halbe Umdrehung beträgt.

Da es für jede Plattenoberfläche einen Lese-Schreib-Kopf gibt, können mehrere Spuren gelesen werden, ohne daß der Arm bewegt werden muß. Der imaginäre Zylinder, der von den Spuren gebildet wird, die übereinander liegen, heißt ein *Zylinder*. Man kann also sagen: Alle Spuren eines Zylinders liegen ohne weitere Armbewegung im Zugriff. Die Zylinderorganisation schließt außerdem, sobald die erste Spur gefunden worden ist, die Rotationsverzögerung aus, denn alle Spuren beginnen an der gleichen Stelle im Zylinder, und das Ende jeder Spur ist gleichzeitig der Anfang jeder Spur.

Die Trommel ist dem Plattenspeicher ähnlich, außer daß sie nur einen einzigen Zylinder mit Spuren enthält, von denen jede ihren eigenen Lese-Schreib-Kopf hat. Dadurch gibt es keine Suchbewegung, und demzufolge auch keine Verzögerung der Suchbewegung. Die Rotationsverzögerung ist minimal wegen der hohen Rotationsgeschwindigkeit der Trommel.

Der Streifenspeicher besteht aus zehn Datenzellen, von denen jede entfern- und auswechselbar ist. Die Datenzelle enthält schmale Magnetbandstreifen; sie befinden sich in 20 Unterzellen mit je 10 Streifen. Die Streifen werden ähnlich wie die Schallplatten in einer Musikbox, angewählt. Der Streifenspeicher dreht sich und bringt die richtige Unterzelle unter eine Zugriffsstation, von der der Streifen heraufgezogen und um eine Trommel gewickelt wird. Da die Trommel rotiert, passiert der Streifen die Lese-Schreib-Köpfe, die ihn lesen oder beschreiben. Ist die Verarbeitung beendet, wird der Streifen wieder an seinen ursprünglichen Platz zurückgelegt.

2.6.4 Datenträger

Der Begriff *Datenträger* (*Volume*) wird benutzt, um sich damit auf eine bestimmte Speichereinheit zu beziehen. Ein Datenträger kann eine Magnetbandrolle, eine Trommel, ein Plattenstapel, eine Unterzelle einer Datenzelle oder der Teil einer 2302 Platteneinheit sein, der von einem Zugriffsmechanismus bedient wird.

2.6.5 Die DD-Karte

Jede Datei, die innerhalb eines Jobsteps benutzt wird, muß mit einer DD-Karte (Data Definition) beschrieben werden. Es ist diese DD-Karte, die dem Betriebssystem sagt, welches E/A-Gerät benutzt werden soll, welche Datenträger mit welchen Datenträgernummern nötig sind, welches der Name der Datei ist und ob diese Datei bereits besteht und gelesen werden soll oder ob sie erzeugt werden soll. Die DD-Karte gibt schließlich die Disposition der Datei an, d.h. sie sagt was mit der Datei geschehen soll, wenn der Jobstep endet. Die folgende DD-Karte ist besonders typisch:

```
//EINDD     DD   DSN=LOHN,DISP=(OLD,KEEP),UNIT=2314,VOL=SER=DA0002
```

Diese DD-Karte heißt EINDD und definiert eine Datei LOHN. LOHN ist eine bereits bestehende Datei (OLD), die nach dem Lesen bestehen

bleiben soll (KEEP). Sie liegt auf einem 2314 Plattenstapel mit der Datenträgernummer DA0002. Würde diese Anweisung nicht auf eine einzige Karte passen, so würde sie auf einer Folgekarte fortgesetzt werden müssen.

Dateien können *katalogisiert* werden, indem man das System anweist, den Dateinamen, die Datenträgernummer und den Typ des E/A-Gerätes festzuhalten. Das erlaubt es, sich auf die Datei durch den Namen allein zu beziehen, ohne daß angegeben werden muß, wo die Datei gespeichert ist. Ist die eben angeführte Datei katalogisiert, so kann die DD-Karte wie folgt geschrieben werden:

```
//EINDD     DD  DSN=LOHN,DISP=(OLD,KEEP)
```

Eine katalogisierte Datei sollte nicht mit einem katalogisierten Verfahren durcheinandergebracht werden. Ein katalogisiertes Verfahren ist ein mit einem Namen versehener Satz von JCL, der in einer besonderen Datei aufbewahrt wird. Für eine katalogisierte Datei wird ihr Name, ihre Datenträgernummer und der Typ ihrer E/A-Einheit vom System festgehalten. So kann es sein, daß die besondere Datei, die die katalogisierten Verfahren enthält, selbst katalogisiert ist.

3. JCL im Job

3.1 Compile-Step

Die JOB-, EXEC- und DD-Karten ermöglichen die Durchführung eines vollständigen Jobs. Als Beispiel stelle man sich vor, daß ein PL/I-Quellenprogramm übersetzt werden soll. Wir nehmen an, daß der PL/I-Übersetzer ein Systemprogramm mit dem Namen PL1COMP ist. Er möge folgende drei Dateien benötigen: Eine Eingabedatei, welche die zu übersetzenden Quellenanweisungen enthält, eine Ausgabedatei zum Drukken der Übersetzungsliste, und eine Ausgabedatei zur Aufnahme des Objektmoduls. (Das Objektmodul enthält die vom Übersetzer erzeugten Instruktionen in Maschinensprache.) Dieses Programm könnte durch die folgende JCL ausgeführt werden.

```
//TEST      JOB 4711,MAIER
//PL1       EXEC PGM=PL1COMP
//SYSPRINT  DD  SYSOUT=A
//COMPOUT   DD  UNIT=SYSDA,DISP=(NEW,PASS),SPACE=(80,250)
//SYSIN     DD  *
  ************************************
  * ANWEISUNGEN DES QUELLEN-PROGRAMMS *
  ************************************
/*
```

Da diese JCL ziemlich abschreckend aussieht, wollen wir jede Karte einzeln betrachten, um herauszufinden, was sie bewirkt. Da eine vollständige Beschreibung aller Parameter in den folgenden Kapiteln gegeben wird, soll die Erläuterung hier nur kurz ausfallen.

```
//TEST      JOB 4711,MAIER
```

Der Job heißt TEST, die Kostenstelle, über die abgerechnet wird, ist 4711, und der Name des Programmierers ist Maier.

```
//PL1       EXEC PGM=PL1COMP
```

Das Programm PL1COMP kommt im Step PL1 zur Ausführung.

```
//SYSPRINT DD  SYSOUT=A
```

SYSOUT=A definiert eine Druckkartei. Druck-und Stanzdateien werden so oft benutzt, daß eine besondere Abkürzung, nämlich SYSOUT existiert. Das Schlüsselwort SYSOUT weist das System an, die Ausgabe auf einen Speicher mit direktem Zugriff in eine Warteschlange zu stellen. Die allgemeine Form ist SYSOUT=klasse, wobei die Klasse von der Installation definiert wird. Traditionsgemäß ist die Ausgabeklasse SYSOUT=A eine Druckausgabe und SYSOUT=B eine Stanzausgabe. Aber natürlich können andere Ausgabeklassen eingerichtet werden, etwa für besondere Formulare oder für besonders umfangreiche Ausgabe etc.

SYSPRINT ist der *ddname* (*Data Definition Name*) der DD-Karte. Das Programm PL1COMP definiert intern eine Datei mit dem ddname SYSPRINT. Diejenigen, die ihre eigenen Programme schreiben, können völlig frei irgendeinen ddname wählen, jedoch muß man die von einem existierenden Programm geforderten ddname wissen.[1])

```
//COMPOUT  DD  UNIT=SYSDA,DISP=(NEW,PASS),SPACE=(80,250)
```

Die COMPOUT DD-Karte beschreibt die Datei der vom Übersetzer erzeugten Objektmodule. Der dabei verwendete UNIT-Parameter spezifiziert das zu benutzende E/A-Gerät.

Es gibt drei Möglichkeiten das E/A-Gerät anzugeben: Die Hardware-Adresse, den Geräte-Typ und die Geräte-Gruppe. Jedes E/A-Gerät der Rechenanlage hat eine ihm eigene *Hardware-Adresse* aus drei Zeichen. Z.B. könnten drei am Rechner angeschlossene Bänder die Hardware-Adressen 0C1, 0C2 und 0C3 besitzen. UNIT=0C2 würde dann das eine bestimmte Bandgerät ansprechen. In einem Multiprogramming-System ist es jedoch besser, statt der Hardware-Adresse, den Geräte-Typ anzugeben, denn die angesprochene Einheit könnte bereits besetzt sein. Unter *Geräte-Typ* versteht man die Einheitenklasse. Die Angabe UNIT=2400-3 z.B. erlaubt den Gebrauch jeder verfügbaren 2400 Modell 3 Bandeinheit. Die *Geräte-Gruppe* schließlich gibt der Installation die Möglichkeit, eine Gruppe von E/A-Einheiten zu einer Gruppe zusammenzuschließen und diese mit einem Namen zu versehen. Die Geräte-Gruppe SYSSQ (System Sequentiell) wird sehr oft benutzt, um E/A-Ge-

[1]) Anmerkung des Übersetzers: Der Programmierer muß die ddname dem Arbeitsvorbereiter, der die JCL zum Programm kodieren soll, mitteilen.

räte zusammenzufassen, die sequentielle Dateien speichern können (also Bandeinheiten und Speicher mit direktem Zugriff). Ähnlich wird die Geräte-Gruppe SYSDA (System Direct Access) benutzt, um Speicher mit direktem Zugriff anzusprechen.

Das Schlüsselwort DISP beschreibt den Status der Datei beim Start des Jobsteps und ihre Disposition nach Abschluß des Steps. NEW zeigt an, daß die Datei in diesem Jobstep erzeugt wird. PASS veranlaßt, daß die Datei zum nächsten Jobstep hinübergereicht wird.

Das Schlüsselwort SPACE=(80,250) fordert Platz für 250 80-stellige Blöcke an. Allen Dateien mit der Disposition DISP=(NEW,...) muß Platz zugeordnet werden, wenn sie auf Speichern mit direktem Zugriff zu liegen kommen sollen.

```
//SYSIN     DD  *
```

Der Stern (*) ist ein besonderes Zeichen, es teilt dem System mit, daß dieser DD-Karte unmittelbar Datenkarten folgen. Das Ende der Eingabedatenkarten wird durch die JCL-Anweisung /* angezeigt.

3.2 Linkage Editor-Step

Als nächstes muß das Programm vom Linkage Editor verarbeitet werden, damit ein ausführbares Lademodul entsteht. Wir wollen annehmen, daß der Linkage Editor ein Systemprogramm ist, das LINKEDIT heißt, und drei Dateien benötigt: Eine Eingabedatei, die die vom Übersetzer erzeugten Objektmodule enthält, eine Ausgabedatei zum Andrucken von Nachrichten und Fehlermeldungen, und eine Ausgabedatei, die das erzeugte Lademodul aufnehmen soll.

```
//TEST      JOB 4711,MAIER
//PL1       EXEC PGM=PL1COMP
//SYSPRINT  DD  SYSOUT=A
//COMPOUT   DD  UNIT=SYSDA,DISP=(NEW,PASS),SPACE=(80,250)
//SYSIN     DD  *
  ************************************
  * ANWEISUNGEN DES QUELLEN-PROGRAMMS *
  ************************************
```

```
/*
//LKED      EXEC PGM=LINKEDIT
//SYSPRINT  DD   SYSOUT=A
//LKEDIN    DD   DSN=*.PL1.COMPOUT,DISP=(OLD,DELETE)
//LKEDOUT   DD   DSN=&&TEMP(GO),DISP=(NEW,PASS),UNIT=SYSDA,
//               SPACE=(1024,(200,20,1))
```

Wir wollen die hinzugefügten Karten im einzelnen besprechen.

```
//LKED      EXEC PGM=LINKEDIT
```

Diese Karte ruft das LINKEDIT Programm zur Ausführung auf.

```
//SYSPRINT  DD   SYSOUT=A
```

Die SYSPRINT DD-Karte definiert die Druckausgabe.

```
//LKEDIN    DD   DSN=*.PL1.COMPOUT,DISP=(OLD,DELETE)
```

Die LKEDIN DD-Karte beschreibt die Eingabedatei, die die Objektmodule für den Linkage Editor enthält, eine Datei, die im vorausgegangenen Step erzeugt worden ist. DSN= *.PL1.COMPOUT sagt dem System, daß dies die Datei ist, die im Step PL1 von der DD-Karte COMPOUT beschrieben wurde. (Die Verwendung von *.step.ddname in einem Parameter heißt ein *Rückbezug*.)

Da diese Datei bereits besteht, ist ihr Status bei Step-Beginn OLD. Die Datei wird nach diesem Jobstep nicht mehr benötigt, man kann also DELETE angeben, um den von ihr belegten Platz auf dem Speicher mit direktem Zugriff freizugeben.

```
//LKEDOUT   DD   DSN=&&TEMP(GO),DISP=(NEW,PASS),UNIT=SYSDA,
//               SPACE=(1024,(200,20,1))
```

Diese DD-Karte ist der COMPOUT DD-Karte vom ersten Step ähnlich, den DSN Schlüsselwortparameter ausgenommen. Die Lademodulausgabe muß als Glied einer untergliederten Datei auf einem Speicher mit direktem Zugriff gespeichert werden. GO ist der Name dieses Gliedes, und die untergliederte Datei heißt TEMP. Die "Unds" && vor dem Dateinamen kennzeichnen die Datei als temporär - d.h. sie wird bei Jobende gelöscht werden. Platz wird für 200 1024-stellige Blöcke angefordert; falls mehr Platz nötig ist, werden zusätzlich 20 weitere 1024-

stellige Blöcke zugeordnet. Die letzte 1 im SPACE-Parameter bedeutet, daß 1 Block reserviert wird, um die Namen der Glieder der unterglie-derten Datei zu speichern (dieser Block enthält 256 Bytes).

3.3 Ausführungs-Step

Das Lademodul ist nun zur Ausführung fertig.

```
//TEST      JOB 4711,MAIER
//PL1       EXEC PGM=PL1COMP
//SYSPRINT  DD  SYSOUT=A
//COMPOUT   DD  UNIT=SYSDA,DISP=(NEW,PASS),SPACE=(80,250)
//SYSIN     DD  *
  ************************************
  * ANWEISUNGEN DES QUELLEN-PROGRAMMS *
  ************************************
/*
//LKED      EXEC PGM=LINKEDIT
//SYSPRINT  DD  SYSOUT=A
//LKEDIN    DD  DSN=*.PL1.COMPOUT,DISP=(OLD,DELETE)
//LKEDOUT   DD  DSN=&&TEMP(GO),DISP=(NEW,PASS),UNIT=SYSDA,
//              SPACE=(1024,(200,20,1))
//GO        EXEC PGM=*.LKED.LKEDOUT
//SYSPRINT  DD  SYSOUT=A
```

Wieder wollen wir die hinzugefügten Karten im einzelnen untersuchen.

```
//GO        EXEC PGM=*.LKED.LKEDOUT
```

Der Rückbezug sagt dem System, daß das auszuführende Lademodul im LKED-Step durch die LKEDOUT DD-Karte beschrieben wurde.

```
//SYSPRINT  DD  SYSOUT=A
```

Die SYSPRINT DD-Karte beschreibt die Druckausgabe.

3.4 Katalogisierte Verfahren

Alle JCL-Karten stehen nun zum Übersetzen, für den Linkage Editor-Lauf und zur Ausführung des Programms zur Verfügung - aber es sind auch sehr viele. Wir sollten aus ihnen nun ein katalogisiertes Verfahren machen, damit mehrere Programmierer diese JCL benutzen können. Man erreicht das, indem man die JCL-Anweisungen als Glied einer untergliederten Datei auf einem Speicher mit direktem Zugriff speichert. (Wie man das macht wird in Kapitel 9 beschrieben.) Ein 1-8 stelliger Name wird für das katalogisierte Verfahren gewählt. PL1CLG[2]) könnte für ein PL/I-Verfahren zum Übersetzen, zum Verarbeiten mit dem Linkage Editor und zum Ausführen passen. Die folgenden Karten bilden das Verfahren.

```
//PL1CLG    PROC
//PL1       EXEC PGM=PL1COMP
//SYSPRINT  DD   SYSOUT=A
//COMPOUT   DD   UNIT=SYSDA,DISP=(NEW,PASS),SPACE=(80,250)
//LKED      EXEC PGM=LINKEDIT
//SYSPRINT  DD   SYSOUT=A
//LKEDIN    DD   DSN=*.PL1.COMPOUT,DISP=(OLD,DELETE)
//LKEDOUT   DD   DSN=&&TEMP(GO),DISP=(NEW,PASS),UNIT=SYSDA,
//               SPACE=(1024,(200,20,1))
//GO        EXEC PGM=*.LKED.LKEDOUT
//SYSPRINT  DD   SYSOUT=A
```

Die JOB-Karte, die SYSIN DD-Karte, die Anweisungen des Quellenprogramms und die /* -Karte wurden weggelassen; sie müssen hinzugefügt werden, wenn der Job abgegeben wird. Die Anweisungen des Verfahrens können aufgerufen werden, indem man den Namen des Verfahrens in der EXEC-Karte angibt.

---

[2]) Anmerkung des Übersetzers: CLG ist die Abkürzung von Compile, Link und Go. Diese Abkürzung paßt sich natürlich nur dem englischen Sprachgebrauch an; sie hat sich aber auch im Deutschen eingebürgert.

```
//TEST      JOB 4711,MAIER
//AUFRUF    EXEC PL1CLG
//PL1.SYSIN DD  *
  ************************************
  * ANWEISUNGEN DES QUELLEN-PROGRAMMS *
  ************************************
/*
```

Das Fehlen des Schlüsselwortes PGM in der EXEC-Anweisung zeigt an, daß ein katalogisiertes Verfahren aufgerufen wird. Der Vorsatz PL1. vor dem ddname SYSIN teilt dem System mit, daß diese DD-Karte für den Step PL1 bestimmt ist. Mit der Verwendung des Verfahrens ist die notwendige JCL handlich geworden.

4. Formate und Regeln für die JCL-Karten

4.1 Format der JCL-Anweisung

Alle JCL-Karten (ausgenommen die /*-Karte) beginnen mit // in den Spalten 1 und 2, ihnen folgt das Namens-, Operations- und Operandenfeld. Das Namensfeld beginnt unmittelbar hinter dem zweiten Schrägstrich, dagegen werden Namens-, Operations- und Operandenfeld voneinander durch eine oder mehrere Leerstellen getrennt.

```
//Name     Operation Operand
```

Das *Namensfeld* identifiziert die JCL-Karte, damit andere Karten oder Kontrollblöcke des Systems auf sie Bezug nehmen können. Der Name kann längenmäßig von 1 bis 8 Stellen variieren, er kann alle alphanumerischen Zeichen (A bis Z, 0 bis 9) und die sogenannten nationalen Zeichen (@ $ #) enthalten. Allerdings muß das erste Zeichen in Spalte 3 stehen.

richtig	falsch	
//A	//+TEST	(1. Zeichen nicht A-Z, 0-9, @ $ #)
//TEST # 10	//RAUMKAPSEL	(mehr als 8 Zeichen)
//EINGABE4	//TEST-6	(der Gedankenstrich ist als Zeichen nicht zugelassen)

Das *Operationsfeld* gibt den JCL-Karten-Typ an: JOB, EXEC, DD, PROC oder Operator-Befehl.

```
//TEST     JOB
//STEP     EXEC
//PRINT    DD
```

Das *Operandenfeld* enthält durch Komma getrennte Parameter. Parameter bestehen aus Schlüsselworten und Variablen, für die man Daten einsetzen muß. Das Operandenfeld hat keine feste Länge und keine Spalteneinteilung, dagegen muß es von mindestens je einem Leerzeichen begrenzt werden.

```
//TEST20   JOB 2095,CLASS=A
//STEP1    EXEC PL1CLG
//PRINT    DD  SYSOUT=A
```

4.2 Parameter im Operandenfeld

Das Operandenfeld wird von 2 Parametertypen aufgebaut:
Dem *Positionsparameter*, der durch seine Stellung im Operandenfeld in bezug auf die anderen Parameter charakterisiert ist, und dem *Schlüsselwortparameter*, positionsmäßig von seinesgleichen unabhängig, der durch das Schlüsselwort, gefolgt von einem Gleichheitszeichen und variabler Information, charakterisiert ist. Das folgende Beispiel enthält beide Typen, den Positionsparameter 3645 und den Schlüsselwortparameter PRTY=1.

```
//TEST     JOB 3645,PRTY=1
```

Der Positionsparameter oder die variable Information in einem Schlüsselwortparameter besteht manchmal aus einer *Unterparameterliste*.
Eine solche Liste kann wieder beide Typen von Parametern, nämlich Positions- und Schlüsselwortparameter, enthalten. Sie folgen den gleichen Regeln und Beschränkungen wie Positions- und Schlüsselwortparameter. Eine Unterparameterliste muß in Klammern eingeschlossen werden, es sei denn, die Liste verkürzt sich auf einen einzelnen Unterparameter. Die folgende JOB-Karte veranschaulicht Positionsunterparameter.

```
//TEST     JOB (3645,100,40),REGION=(100K,400K)
```

Alle Unterparameter sind Positionsunterparameter; (3645, 100, 40) ist ein Positionsparameter, REGION=(100K,400K) ist ein Schlüsselwortparameter. Unterparameter können aber auch aus Schlüsselwortparametern bestehen. DCB=(LRECL=80, BLKSIZE=1600) stellt einen Schlüsselwortparameter dar, der aus Schlüsselwortunterparametern besteht.

4.3 Regeln für Parameter

1) Beide Positions- und Schlüsselwortparameter müssen durch Kommata getrennt werden, Leerstellen sind nicht erlaubt.

richtig	falsch
6,PRTY=1,REGION=104K	6, PRTY=1,REGION=104K (die Leerstelle ist nicht zulässig)
(6,106),PRTY=1	(6,106)PRTY=1 (das Komma fehlt)
PRTY=1,REGION=104K	PRTY=1,REGION=104K, (das letzte Komma ist zuviel)

2) Positionsparameter müssen im Operandenfeld in der von der IBM vorgeschriebenen Folge vor allen Schlüsselwortparametern kodiert werden.

richtig	falsch
(6,106),PRTY=1	PRTY=1,(6,106)

3) Das Fehlen eines Positionsparameters wird durch das Kodieren eines Auslassungskommas angezeigt. Der 2. Positionsparameter soll in der folgenden JOB-Karte weggelassen werden.

```
//TEST      JOB (3645,100,40),MSGLEVEL=(1,1)
```

```
//TEST      JOB (3645,,40),MSGLEVEL=(1,1)
```

4) Auslassungskommas müssen nicht kodiert werden, falls der folgende Positionsparameter der letzte ist oder falls alle folgenden Positionsparameter auch fehlen.

```
//TEST      JOB (3645,,),MSGLEVEL=(1,1)
```

Statt dessen kann kodiert werden:

```
//TEST      JOB (3645),MSGLEVEL=(1,1)
```

5) Besteht ein Unterparameter aus einem einzigen Wert, so können die einschließenden Klammern weggelassen werden.

```
//TEST      JOB (3645),MSGLEVEL=(1,1)
```

Statt dessen kann kodiert werden:

```
//TEST      JOB 3645,MSGLEVEL=(1,1)
```

6) Fehlen alle Positionsparameter, so muß nichts kodiert werden.

```
//TEST      JOB MSGLEVEL=(1,1)
```

7) Schlüsselwortparameter können im Operandenfeld nach den Positionsparametern in beliebiger Reihenfolge kodiert werden.

```
//TEST      JOB 3645,PRTY=1,MSGLEVEL=(1,1),REGION=104K
//TEST      JOB 3645,MSGLEVEL=(1,1),REGION=104K,PRTY=1
```

Diese Regeln erscheinen umfangreich, aber ein wenig Praxis läßt ihre Anwendung zur Routine werden. Die folgenden Beispiele sollen helfen, die Regeln einzuprägen.

```
//A         DD  DCB=(RECFM=F,LRECL=80),
//              SPACE=(TRK,(100,80,30),RLSE,CONTIG,ROUND)
```

Die obige DD-Karte paßt nicht auf eine einzige Karte und wird deshalb auf einer zweiten fortgesetzt. Die Regeln für die Fortsetzung werden im nächsten Abschnitt besprochen. Die folgenden DD-Karten zeigen wie die Karten kodiert werden müssen, wenn die unterstrichenen Parameter weggelassen werden.

```
//A         DD  DCB=(LRECL=80,RECFM=F),
//              SPACE=(TRK,(100,,30),RLSE,,ROUND)
              UND
//A         DD  DCB=LRECL=80,SPACE=(TRK,100,RLSE)
              UND
//A         DD  SPACE=(TRK,100)
```

4.4 Allgemeine Regeln

1) Man beginnt bei allen Karten in Spalte 1 mit // bzw. /*.

2) Ein Eintrag im Namensfeld (der manchmal wahlfrei ist) muß in Spalte 3 beginnen, ihm muß wenigstens eine Leerspalte folgen.

```
//LAUF#12   JOB
```

3) Die Felder müssen durch wenigstens eine Leerspalte voneinander getrennt werden.

```
//STEP1    EXEC PL1CLG
//STEP1      EXEC   PL1CLG
```

4) In die Felder dürfen keine Leerstellen eingeschoben werden, die Parameter müssen durch Kommata getrennt werden.

5) Kommentare können auf jede beliebige JCL-Karte geschrieben werden, indem man eine oder mehrere Leerstellen zwischen dem letzten Feld und dem Kommentaranfang läßt.

```
//STEPL    EXEC PL1CLG        DAS IST EIN KOMMENTAR
```

6) Die Spalten 1 bis 71 werden für JCL und Kommentare benutzt. Falls mehr als eine Karte nötig wird, oder falls man die Parameter getrennt auf einzelne Karten kodieren möchte, unterbricht man die Anweisung nach einem vollständigen Parameter[1]) (das ihm folgende Komma gehört dazu), spätestens aber in Spalte 71[2]). Man kodiert // im Spalte 1 und 2 der Folgekarte und setzt die unterbrochene Anweisung irgendwo zwischen Spalte 4 und 16 fort.

```
//TEST     JOB 6245,MSGLEVEL=(1,1),MSGCLASS=A,
// REGION=104K
                ODER
//TEST     JOB 6245,
// MSGLEVEL=(1,1),MSGCLASS=A,
// REGION=104K
```

7) Felder, die Großbuchstaben bzw. Sonderzeichen enthalten - PARM (, / . , etc. - werden genauso kodiert wie angegeben. Felder mit kleinen Buchstaben müssen mit vom Benutzer gewählten Werten ausgefüllt werden. TIME=minuten z.B. könnte als TIME=10 kodiert werden, um eine Zeitbegrenzung von 10 Minuten zu verlangen.

---

[1]) Die Abrechnungsinformation der JOB-Karte, die Parameter ACCT und PARM der EXEC-Karte, der COND-Parameter der JOB- und EXEC-Karte und die Parameter DCB, VOL=SER, SEP und UNIT=SEP der DD-Karte können auch nach einem vollständigen Unterparameter unterbrochen werden.

[2]) Folgekarten mußten anfangs durch das Ablochen irgendeines Zeichens in die Spalte 72 angezeigt werden; das ist heute nicht mehr nötig, darf aber trotzdem gemacht werden.

8) Kursiv Gedrucktes kann wahlweise kodiert werden. Wenn wir uns die Regeln für das Weglassen unnötiger Kommas und Klammern in Erinnerung rufen, könnte

SPACE=(TRK,(*menge, zusatzmenge*), *RLSE*)

in den folgenden Kombinationen - 1 für menge, 2 für zusatzmenge angenommen - kodiert werden.

SPACE=(TRK,(1,2),RLSE)

SPACE=(TRK,(1,),RLSE) oder SPACE=(TRK,1,RLSE)

SPACE=(TRK,(1,),) oder SPACE=(TRK,1)

SPACE=(TRK,(1,2),) oder SPACE=(TRK,(1,2))

9) Sind die Werte über einer gestrichelten Linie mehrzeilig gestaffelt angeordnet, so bedeutet das, daß einer dieser Werte gewählt werden muß. Sind die Werte kursiv gedruckt, so sind sie wahlweise zu kodieren. Ist einer der gestaffelten Werte unterstrichen, so handelt es sich um die Standardunterstellung.

NEW KEEP

OLD DELETE

DISP=(---,--------)

Dieses Beispiel kann kodiert werden als DISP=(NEW,KEEP), DISP=(OLD,KEEP) etc.. DISP=(,DELETE) ist mit DISP=(NEW, DELETE) gleichbedeutend, weil NEW die Standardunterstellung ist. Da beide Positionsparameter wahlweise kodiert werden können, kann man auch DISP=(,KEEP),DISP=OLD etc. schreiben.

10) Einige Parameter beziehen sich nur auf besondere Systeme oder Konfigurationen. Passen sie für das vorliegende System nicht, so werden sie ignoriert.

5. Beschreibung der JOB-Karte

Die JOB-Karte zeigt dem Betriebssystem OS (Operating System) den Beginn des Jobs an, sie gibt die notwendigen Abrechnungsinformationen und stellt Parameter für den Lauf zur Verfügung. Jeder Job muß mit einer einzelnen JOB-Karte beginnen. Die Installation schreibt gewöhnlich ein festes JOB-Kartenformat vor. Einige in diesem Kapitel beschriebene JOB-Kartenparameter können darin sogar verboten sein.

Die JOB-Karte hat die folgende Form.

```
//jobname    JOB      (kostenstellennummer, abrechnungsinformationen),
//                    name, schlüsselwortparameter
```

jobname - Der dem Job vom Benutzer gegebene Name.
kostenstellennummer, abrechnungsinformationen -

Nummer der Kostenstelle, die mit dem Job belastet werden soll, sowie alle weiteren, in der Installation eingeführten Abrechnungsinformationen.

name - Ein vom Benutzer gewählter Name um den Lauf zu kennzeichnen.
schlüsselwortparameter - Die folgenden Schlüsselwortparameter können in der JOB-Karte kodiert werden.

MSGLEVEL	Gibt an, ob alle JCL-Karten angedruckt werden sollen oder nicht.
PRTY	Gibt die Jobpriorität an (nur MFT und MVT).
MSGCLASS	Gibt für den Job Scheduler die Ausgabeklasse an (nur MFT und MVT).
CLASS	Gibt die Job-Klasse an (nur MFT und MVT).
TYPRUN	Kennzeichnet den Job als HOLD-Job in der Eingabe-Queue (nur MFT und MVT).

Die folgenden Schlüsselwortparameter sind wahlfrei, sie können in der JOB- und EXEC-Karte kodiert werden; ihre Diskussion wird bis Kapitel 7 zurückgestellt.

TIME	Erlegt dem Job eine Zeitbegrenzung auf (nur MVT).
COND	Gibt Bedingungen für die Ausführung nachfolgender Jobsteps an für den Fall, daß vorausgehende Steps abbrechen.
ROLL	Gibt die Rollout/Rollin-Bedingungen an (nur MVT).
REGION	Gibt die Größe der dem Job zuzuordnenden Region an (nur MVT).
RD	Stellt eine Forderung für die Checkpoint/Restart-Einrichtung, die den Wiederanlauf betrifft.
RESTART	Liefert einen Job an, der wieder anlaufen soll.

Der Parameter Jobname und der Operand JOB sind stets erforderlich, Kostenstellennummer und Abrechnungsinformationen sind gewöhnlich zwingend, aber auch andere Parameter können in einer Installation zwingend vorgeschrieben werden.

5.1 Der Jobname: jobname

Der Jobname wird vom Benutzer gewählt. Er identifiziert den Job dem OS gegenüber. Der Jobname kann aus einem bis zu acht alphamerischen Zeichen (A-Z, 0-9) oder sogenannten nationalen Zeichen (@ $#) gebildet werden; das erste muß in Spalte 3 beginnen. In MFT- oder MVT-Systemen sollten die Jobs unterschiedliche Namen haben, da der Operator sie nur durch ihren Namen ansprechen oder auch aus dem System entfernen (Cancel) kann. Unterschiedliche Jobnamen lassen sich erreichen, indem man unterschiedliche Teile, wie z.B. Initialen oder Personalnummern in den Jobnamen mit hineinkodiert.

Die folgenden Namen sollten nicht als Jobnamen benutzt werden, weil sie Operatorbefehle darstellen[1]):

[1]) Falls doch einer dieser Namen als Jobname benutzt wird, wird der Job zwar laufen, aber der Operator muß diesen Jobnamen in Hochkommas einschließen, falls er ihn in einem Befehl benutzt. Z.B. DISPLAY 'JOBNAMES'.

A	JOBNAMES	R
CONSOLES	N	SPACE
DSNAME	Q	STATUS
T		

richtig	falsch	
//GDB406	//+J	(das erste Zeichen ist kein A-Z 0-9, @ $ #)
//A8462	//KOMPLIKATION	(mehr als 8 Zeichen)
//B8750#12	// LAUF 6	(der Name beginnt nicht in Spalte 3)
//X	//Q	(Q ist ein ungültiger Name)

5.2 Abrechnungsinformationen

Die Abrechnungsinformation wird kodiert als (*kostenstellennummer, abrechnungsinformationen*). Die Kostenstellennummer und die Abrechnungsinformationen müssen in ihrem Aufbau durch die Installation vorgegeben werden. Unterparameter müssen durch Kommas getrennt werden.

Die Kostenstellennummer ist wahlfrei, sie kann jedoch als zwingend vorgeschrieben werden und zwar bei einem PCP-System, wenn es generiert wird, und bei einem MFT- oder MVT-System in den katalogisierten Reader-Prozeduren. Die Gesamtzahl der Zeichen für die Kostenstellennummer und die Abrechnungsinformationen (die Kommas, die die Unterparameter trennen, eingeschlossen) darf 142 nicht überschreiten. Enthalten die Kostenstellennummer oder die Abrechnungsinformationen irgendein Sonderzeichen außer einem Gedankenstrich (also Leerstelle ,./'()*&+=), so muß der Unterparameter in Anführungsstriche (') eingeschlossen werden. Die Anführungsstriche sind kein Teil der Information. Ein Anführungsstrich als Teil einer Information muß durch zwei aufeinanderfolgende Anführungsstriche kodiert werden; z.B. muß D'DORF als D''DORF kodiert werden.

Es gelten die allgemeinen Regeln für das Weglassen von Kommas und Klammern in Unterparameterlisten. Wenn die Kostenstellennummer

weggelassen wird, muß ihre Abwesenheit durch ein Komma angezeigt werden: (,abrechnungsinformationen). Fehlen dagegen die Abrechnungsinformationen, so können die Klammern um die Kostenstellennummer fortfallen. Die Abrechnungsinformationen können nach einem vollständigen Unterparameter, das Komma eingeschlossen, auf einer Folgekarte fortgesetzt werden.

```
//JOBNAME  JOB 2096
//JOBNAME  JOB (20746,30,
//              6,94)
//JOBNAME  JOB 3042,6
```

falsch! Bei mehr als einem Positionsunterparameter sind Klammern nötig.

Statt in Klammern kann man die Abrechnungsinformation in Hochkommas einschließen. Dadurch erübrigt es sich, Unterparameter, die Sonderzeichen enthalten, in Hochkommas einzuschließen, doch muß in diesem Fall die Abrechnungsinformation vollständig auf eine Karte kodiert werden.

```
//JOBNAME  JOB '2256/2240'
```

5.3 Programmierername: name

Das Feld name ist ein vom Benutzer gewählter 1-20 Zeichen langer Name, um sich oder den Job zu kennzeichnen. Enthält das Feld außer dem Punkt Sonderzeichen (wie Leerstellen,/'()*&+-=), so muß es in Hochkommas eingeschlossen werden. Ein Hochkomma im Feld muß durch zwei aufeinanderfolgende Hochkommas kodiert werden; O'CLOCK wird kodiert als 'O''CLOCK'.

Der Parameter name kann für eine Installation zwingend vorgeschrieben werden und zwar bei einem PCP-System, wenn es generiert wird, und bei einem MFT- oder MVT-System in den katalogisierten Reader-Prozeduren.

```
//JOB1     JOB (4562,200,10),SCHMITT
//JOB2     JOB ,A.SCHMITT
```

Fehlt die Abrechnungsinformation, so muß ihr Fehlen durch ein Auslassungskomma angezeigt werden.

```
//JOB3      JOB 6542,'O''CLOCK'
//JOB4      JOB 5642,AL GOLL
```

Falsch! Da das Feld name ein anderes Sonderzeichen als den Punkt enthält, muß es in Hochkommas eingeschlossen werden.

5.4 Andrucken der JCL-Karten: MSGLEVEL

MSGLEVEL= (*jcl, zuordnung*) wird kodiert, um das Andrucken der JCL-Karten und der Zuordnungsmeldungen zu steuern. Wenn verlangt, erscheinen die *Zuordnungsmeldungen* (Allocation Messages) am Anfang des Jobsteps, um die Zuordnung der Dateien zu den E/A-Geräten zu zeigen, und am Ende des Steps, um die Disposition der Dateien anzugeben. Wird MSGLEVEL nicht kodiert, so tritt eine Standardunterstellung in Kraft. Diese Unterstellung wird beim PCP-System bei der Generierung, beim MFT- und MVT-System in den katalogisierten Reader-Prozeduren gesetzt. Die von IBM zur Verfügung gestellten Reader-Prozeduren unterstellen MSGLEVEL=(0,1).

Die Werte für jcl und zuordnung sind wie folgt:

jcl	Bedeutung
0	Nur die JOB-Karte wird gedruckt
1	Die JCL des Eingabestromes und alle aufgerufenen katalogisierten Verfahren werden angedruckt. Die Darstellung der Anweisungen nach dem Ersetzen der symbolischen Parameter ist mit eingeschlossen.
2	Nur die JCL des Eingabestroms wird angedruckt.

zuordnung	Bedeutung
0	Es werden keine Zuordnungsmeldungen gedruckt, es sei denn, der Job terminiert abnormal.
1	Alle Zuordnungsmeldungen werden gedruckt.

```
//TEST      JOB 6524,'PETER F. MAIER',
//              MSGLEVEL=(1,0)
```

Alle JCL, aber keine Zuordnungsmeldungen werden gedruckt.

```
//JOB1    JOB 5540,STARK,MSGLEVEL=2
```

Nur die JCL des Eingabestroms wird gedruckt. Das Drukken der Zuordnungsmeldungen hängt vom unterstellten Wert ab.

```
//JOB2    JOB 4422,STARK,MSGLEVEL=(,1)
```

Das Drucken der JCL-Karten hängt vom unterstellten Wert ab. Die Zuordnungsmeldungen werden angedruckt.

MSGLEVEL=(o,o) darf nicht unterstellt noch kodiert werden, wenn in MFT oder MVT die Restart-Einrichtung, oder wenn in MVT die System Management-Einrichtung benutzt wird.

JCL-Karten aus dem Eingabestrom werden mit // in den Spalten 1 und 2, Kommentar-Karten (//*) mit *** in den Spalten 1 bis 3, und Karten aus katalogisierten Verfahren mit X X in den Spalten 1-2 angedruckt. Jede DD-Karte, die ein katalogisiertes Verfahren überschreibt, wird mit X/ in den Spalten 1-2 angedruckt. Enthält eine JCL-Karte nur Kommentar, so wird sie mit X X* in den Spalten 1 bis 3 angelistet.

5.5 Jobpriorität: PRTY (nur MFT und MVT)

PRTY=*priorität* gibt die *Job-Initialisierungs-Priorität* an, das ist die Priorität, mit der die Jobs aus der Queue ausgewählt werden, um ausgeführt zu werden. Die Priorität rangiert von 0 (niedrigste) bis 13 (höchste). Wird PRTY weggelassen, so wird ein in der Reader-Prozedur angegebener Wert unterstellt. Die Priorität 13 ist für das System vorgesehen und sollte nicht verwendet werden.

Die Priorität wirkt innerhalb der Jobklassen. Sind mehrere Jobs einer gegebenen Klasse in der Queue und warten dort auf ihre Initialisierung, so wird der Job mit der höchsten Priorität innerhalb der Klasse zuerst ausgewählt. In MVT bestimmen die Initiator/Terminators und in MFT die Partitions welche Jobklassen für die Jobauswahl die höchste Priorität haben. Die *Dispatching-Priorität* ist die Priorität, mit der ein Job im Speicher die CPU zugeteilt bekommt. Sie hängt im MVT von der

Jobpriorität, im MFT von der Partition ab. Partition 0 hat die höchste Priorität, Partition 51 die niedrigste. (Die Dispatching-Priorität kann mit dem Parameter DPRTY der EXEC-Karte gesetzt werden.)

MVT-Systeme mit der Einrichtung Time-Slicing werden so generiert, daß alle Jobs einer gegebenen Priorität Time-Slicing erfahren. Auf diese Art und Weise verursacht der PRTY-Parameter, daß ein MVT-Job Time-Slicing erfährt. (Im MFT wird der CLASS-Parameter benutzt, um die Jobs zusammenzufassen, die Time-Slicing erfahren sollen.)

```
//JOB3      JOB (2252,40,10),'PETER F. MAIER',
//                MSGLEVEL=1,PRTY=6
```

5.6 Systemnachrichten: MSGCLASS (nur MFT und MVT)

MSGCLASS= *klasse* gibt die Ausgabeklasse für die Meldungen des Job Schedulers an. MFT- und MVT-Systeme schreiben die Druck- und Stanzausgabe zuerst auf einen Speicher mit direktem Zugriff. Später geben Writer-Prozeduren die Ausgabedaten auf Drucker und Stanzer aus. Die Ausgabeklasse ist ein einzelnes Zeichen, nämlich A-Z oder 0-9. MSGCLASS=A wird unterstellt, falls der Parameter fehlt. Diese Klasse ist traditionsgemäß ein Drucker.

Die *Job Scheduler-Meldungen* beinhalten alle Meldungen, die nicht vom Jobstep gedruckt werden, also JCL-Karten, Fehlernachrichten, Zuordnungsmeldungen, Datei-Dispositions-Meldungen und Abrechnungsinformationen. Da der Writer eines MFT- oder MVT-Systems vom Operator gestartet werden muß, sollte man eine aktive Ausgabeklasse benutzen, weil die Ausgabe sonst nicht ausgedruckt werden würde, sondern zwischengespeichert bliebe.

Für sehr umfangreiche Ausgaben und für Ausgaben auf besonderen Formblättern oder nur um eine bestimmte Ausgabe abzutrennen, sollte man eigene Ausgabeklassen benutzen. (Die Ausgabedateien einer bestimmten Klasse werden zusammen gedruckt oder gestanzt.) Man wird selten Anlaß haben, für die Job Scheduler-Meldungen eine besondere Ausgabeklasse zu verwenden.

```
//JOB7      JOB 2564,SCHMITT,PRTY=2,MSGCLASS=V
```

5.7 Jobklasse: CLASS (nur MFT und MVT)

CLASS=*klasse* gibt die Jobklasse an. Jede Installation muß bestimmen, welche von den 15 Jobklassen benutzt werden sollen. Die Jobklassen rangieren von A bis O. Fehlt CLASS, so wird CLASS=A unterstellt. Der Operator eines MVT-Systems startet einen Initiator/Terminator für jede Jobklasse. In MFT-Systemen sind die Jobklassen den Partitions zugeordnet. Die Zuordnung geschieht bei der Generierung, sie kann aber vom Operator verändert werden. Einer Klasse muß eine Partition (MFT) oder ein Initiator/Terminator (MVT) zugeordnet sein; ist das nicht der Fall, so bleibt der Job in der Queue und läuft nicht.

In der Installation versucht man die Jobklassen so einzurichten, daß ein Gleichgewicht zwischen E/A- und rechen-intensiven Jobs, zwischen großen und kleinen Jobs, etc., besteht. In der Praxis läßt sich das nur schwer erreichen, teilweise deshalb, weil der Programmierer nicht immer sagen kann, ob sein Job E/A- oder rechen-intensiv ist. Trotzdem ist der Versuch der Jobklassifikation die Anstrengung wert, denn die Leistungsfähigkeit von Multiprogramming-Systemen hängt stark von einem ausgewogenen Jobmix ab.

Die Jobklasse bestimmt zusammen mit den Parametern PRTY und DPRTY die Gesamtpriorität des Jobs. Der Operator kann die verschiedenen Jobklassen nach seinem Gutdünken starten und stoppen, und bestimmt damit, wann bestimmte Jobklassen laufen.

MFT-Systeme mit der Time-Slicing-Einrichtung werden so generiert, daß alle Jobs bestimmter Klassen Time-Slicing gemeinsam erfahren. Auf diese Art und Weise bestimmt der CLASS-Parameter, wann ein MFT-Job Time-Slicing erfährt. (In MVT wird der Parameter PRTY benutzt, um Jobs, die dem Time-Slicing unterworfen werden sollen, zu klassifizieren.)

```
//TERMINAL JOB 6655,JOHNSON,MSGCLASS=W,CLASS=B
```

5.8 HOLD-Job in der Eingabe-Queue: TYPRUN (nur MFT und MVT)

TYPRUN=HOLD hält einen Job in der Eingabe-Queue fest, damit er erst später ausgeführt wird. Der Job wird solange festgehalten, bis der Ope-

rator den Befehl RELEASE gibt. Unterrichten Sie den Operator unbedingt über einen Job, der in der Queue festgehalten wird; beim Einlesen des Jobs wird nämlich keine Meldung ausgegeben, und wenn der Operator nicht nachsieht, merkt er nicht, daß ein Job in der Queue gehalten wird, der noch ausgeführt werden muß.

Die Anweisung TYPRUN=HOLD wird verwendet, wenn man erreichen will, daß ein Job erst nach der Beendigung der anderen Jobs läuft. Da MVT- und MFT-Systeme nicht leicht zu bedienen sind, sollte man die Operatoreingriffe so gering wie möglich halten.

```
//X15       JOB (2001,10),'J.K.L.',CLASS=M,TYPRUN=HOLD
```

5.9 Zusammenfassung

Jeder Job muß mit einer JOB-Karte beginnen. Verschiedene JOB-Parameter können für die betreffende Installation als zwingend vorgeschrieben werden. Die einfachste Form der JOB-Karte ist:

```
//JOBNAME   JOB
```

Die bisher behandelten verschiedenen JOB-Karten-Parameter sind:

```
//jobname JOB (kostenstellennummer, abrechnungsinformationen), name,
//              MSGLEVEL=(jcl, zuordnung), PRTY=priorität,
//              MSGCLASS= klasse, CLASS=klasse, TYPRUN=HOLD
```

Da die Parameter (*kostenstellennummer, abrechnungsinformationen*) und *name* Positionsparameter sind, muß ein Komma ihr Fehlen anzeigen, es sei denn, beide Parameter fehlen.

```
//LAUF1     JOB ,SCHMITT,MSGLEVEL=(1,1)
//LAUF2     JOB (2465,33),,MSGLEVEL=(1,1)
//LAUF3     JOB MSGLEVEL=(1,1)
```

Die folgenden Beispiele zeigen verschiedene JOB-Karten, wie sie oft benutzt werden.

```
//TEST23A   JOB (6500,100,50),SCHMITT,MSGLEVEL=1,
//              CLASS=L,PRTY=8
```

Der Job TEST23A von Herrn Schmitt wird ausgeführt und der Kostenstelle 6500 belastet. Die Angaben 100 und 50 sind Werte für die in dieser Installation definierten Abrechnungsparameter, etwa geschätzte Angaben für die Druck- und Stanzausgabe. Die Jobklasse ist L, die Priorität beträgt 8, und alle JCL-Karten werden angedruckt.

```
//DDE       JOB 4524,JONES,MSGCLASS=V,TYPRUN=HOLD
```

Der Job DDE von Herrn Jones wird der Kostenstelle 4524 belastet. Die Systemnachrichten werden in der gesonderten Ausgabeklasse V angedruckt. Der Job wird solange in der Eingabe-Queue festgehalten, bis er durch einen Operatorbefehl freigegeben wird.

6. Beschreibung der EXEC-Karte

Jeder Jobstep beginnt mit einer EXEC-Karte, die das auszuführende Programm (Compiler, Linkage Editor, Anwendungsprogramm, Utility, etc.) nennt oder die ein katalogisiertes Verfahren aufruft. Ein katalogisiertes Verfahren kann selbst mehrere Jobsteps enthalten, von denen jeder mit einer EXEC-Karte beginnt, die das auszuführende Programm nennt. In einem Job können viele EXEC-Karten (maximal 255) enthalten sein, denen DD-Karten folgen, die die vom Jobstep geforderten Dateien beschreiben.

Das Format der EXEC-Karte ist wie folgt:

```
                    prozedur
                    PROC=prozedur
                    PGM=programm
                    PGM=*.rückbezug
//stepname EXEC ---------------, schlüsselwortparameter
```

stepname - Der für den Jobstep vom Benutzer gewählte Name.
rückbezug - Der Name einer vorausgehenden DD-Karte, die das auszuführende Programm beschreibt.
program - Das auszuführende Programm.
prozedur - Das zu benutzende katalogisierte Verfahren.
schlüsselwortparameter - Die folgenden Schlüsselwortparameter können in der EXEC-Karte kodiert werden:

PARM	Übergibt Parameter an den Jobstep.
ACCT	Stellt Abrechnungsinformationen für den Jobstep zur Verfügung.
DPRTY	Setzt die Dispatching - Priorität des Jobsteps (nur MVT).

Die folgenden Schlüsselwortparameter sind wahlfrei, sie können in der JOB- und EXEC-Karte kodiert werden; ihre Diskussion wird bis Kapitel 7 zurückgestellt.

TIME	Erlegt dem Jobstep eine Zeitbegrenzung auf (nur MVT).
COND	Gibt Bedingungen für die Ausführung nachfolgender Jobsteps an für den Fall, daß vorausgehende Steps abbrechen.
ROLL	Gibt die Rollout/Rollin-Bedingungen an (nur MVT).
REGION	Gibt die Größe der dem Jobstep zuzuordnende Region an (nur MVT).
RD	Stellt eine Forderung für die Checkpoint/Restart-Einrichtung, die den Wiederanlauf betrifft.

6.1 Der Jobstepname: stepname

// *stepname* EXEC benennt den Jobstep. Der Stepname ist wahlfrei, er besteht aus 1-8 alphamerischen (A bis Z, 0 bis 9) oder sogenannten nationalen Zeichen (@ $ #), die den vom Benutzer gewählten Namen bilden; er muß in Spalte 3 beginnen. Die Stepnamen innerhalb eines Jobs oder eines katalogisierten Verfahrens müssen unverwechselbar sein. Der Stepname ist erforderlich, falls spätere JCL-Anweisungen sich auf diesen Step rückbeziehen, oder falls die EXEC-Karte in einem katalogisierten Verfahren steht.

```
//STEP10   EXEC
//ERRECHNE EXEC
//(GO      EXEC
```

Falsch: Das erste Zeichen ist kein A bis Z, 0 bis 9, @ $ #.

6.2 Der Programmname: programm

// *stepname* EXEC PGM=*programm* benennt das auszuführende Programm. Programme können sich in der Systembibliothek SYS1.LINKLIB, in temporären Bibliotheken und in privaten Bibliotheken befinden. Ein Programm ist ein Lademodul und muß das Glied einer untergliederten Datei sein, die sich auf einem Speicher mit direktem Zugriff befindet.

6.2.1 Programme in der Systembibliothek

Eine Systembibliothek mit dem Namen SYS1.LINKLIB enthält alle von der IBM zur Verfügung gestellten Systemprogramme, wie Compiler, Linkage Editor und Dienstprogramme. Man kann seine eigenen Programme zur SYS1.LINKLIB hinzufügen, solange die gewählten Namen nicht mit den Namen der IBM-Programme in Konflikt kommen. (Wird ein Programm zur SYS1.LINKLIB hinzugefügt, dessen Name gleich dem Namen eines bestehenden Programms ist, so wird das bestehende Programm ersetzt.)

Da die SYS1.LINKLIB alle von der IBM zur Verfügung gestellten Programme enthält, ist sie die wohl wichtigste Systemdatei. Man sollte eigene Programme nicht wahllos zu ihr hinzufügen, weil die Gefahr besteht, daß diese Bibliothek voll wird, und weil man ein bestehendes Programm unbeabsichtigt zerstören kann. Nur voll ausgeprüfte und oft benutzte Programme sollten zur SYS1.LINKLIB hinzugefügt werden. Ein Systemprogramm der SYS1.LINKLIB wird ausgeführt, indem man seinen Namen in der EXEC-Karte kodiert:

```
//STEP1    EXEC PGM=IEFBR14
```

Fehlen andere Anweisungen, so sucht das System in der SYS1.LINKLIB nach dem genannten Programm. IBM-Programme haben Namen, die dem oben angeführten ähnlich sind. Damit wird die Wahrscheinlichkeit, daß jemand einen solchen Namen verwendet, herabgesetzt.

6.2.2 Programme in privaten Bibliotheken

Private Programmbibliotheken werden als Ausgabe des Linkage Editors erzeugt. Kapitel 13 zeigt, wie man diese Ausgabedatei des Linkage Editor als private Bibliothek anlegt. Die Programme in einer Bibliothek müssen unterschiedliche Namen haben; Programme in verschiedenen Bibliotheken dagegen können gleiche Namen haben. Ein privates Programm wird durch den Einschluß einer besonderen JOBLIB DD-Karte, die unmittelbar nach der JOB-Karte folgt, ausgeführt. (Die Parameter der DD-Karte werden in den späteren Kapiteln ausführlich beschrieben.)

```
//TEST     JOB 6245,SCHMITT
//JOBLIB   DD  DSN=COMPLIB,DISP=SHR
//GO       EXEC PGM=COMP1
```

6.2.3 Die JOBLIB DD-Karte

Die oben kodierte JOBLIB DD-Karte definiert COMPLIB als eine private Bibliothek, die das Programm COMP1 enthält. Die JOBLIB Karte wird unmittelbar hinter die JOB-Karte gelegt, und ist für jeden Jobstep wirksam. Sie darf in kein katalogisiertes Verfahren gelegt werden. Wird das Programm in der beschriebenen Bibliothek nicht gefunden, so durchsucht das System die SYS1.LINKLIB. Werden mehrere Programme von verschiedenen Bibliotheken ausgeführt, so müssen die Bibliotheken aneinandergehängt werden (Concatenation).

```
//TEST     JOB 6245,SCHMITT
//JOBLIB   DD  DSN=COMPLIB,DISP=SHR
//         DD  DSN=PRINTLIB,DISP=SHR
```

PRINTLIB wurde durch das Weglassen des ddname in der DD-Karte an COMPLIB angehängt. Das System behandelt COMPLIB und PRINTLIB als wären sie eine Datei.

```
//STEP1    EXEC PGM=COMP1
```

COMP1 ist ein in der COMPLIB enthaltenes Programm.

```
//STEP2    EXEC PGM=PRINTX
```

PRINTX könnte in der PRINTLIB sein.

```
//STEP3    EXEC PGM=IEA001
```

IEA001 könnte ein Systemprogramm sein, das in der SYS1.LINKLIB enthalten ist.

Wenn die Bibliothek nicht katalogisiert ist, d.h. wenn ihr Name und der Platz, an dem sie gespeichert ist, nicht vom System aufgezeichnet sind, muß man die Einheit und die Datenträgernummer des Speichers mit direktem Zugriff, auf dem sie gespeichert ist, mit angeben.

```
//JOBLIB   DD  DSN=COMPLIB,DISP=SHR,UNIT=2311,VOL=SER=DA0007
```

COMPLIB ist nicht katalogisiert und liegt auf dem 2311-Plattenstapel DA0007.

6.2.4 Die STEPLIB DD-Karte

Die STEPLIB DD-Karte, die in Form und Funktion der JOBLIB Karte ähnlich ist, wird hinter die EXEC-Karte gelegt und ist nur für diesen Jobstep wirksam. STEPLIB stellt eine zweite Möglichkeit für die Angabe von privaten Bibliotheken zur Verfügung.

```
//TEST     JOB 6245,SCHMITT
//STEP1    EXEC PGM=COMP1
//STEPLIB  DD  DSN=COMPLIB,DISP=SHR
//STEP2    EXEC PGM=PRINTX
//STEPLIB  DD  DSN=PRINTLIB,DISP=SHR
```

Schließt ein Job beide, nämlich die JOBLIB und die STEPLIB DD-Karte ein, so überschreibt die STEPLIB Karte für ihren Step die JOBLIB Karte. Um die Auswirkung der JOBLIB Karte für einen bestimmten Step aufzuheben, muß man die SYS1.LINKLIB mit einer STEPLIB Karte ansprechen.

Wie für die JOBLIB Karte kann auch in der STEPLIB Karte Concatenation (das Aneinanderhängen mehrerer privater Bibliotheken) verwendet werden. Ist die Bibliothek nicht katalogisiert, so müssen auch hier die Parameter UNIT= und VOL=SER= angegeben werden. Im Gegensatz zur JOBLIB Karte kann die STEPLIB Karte in ein katalogisiertes Verfahren eingefügt werden. Das folgende Beispiel zeigt die Verwendung beider, der JOBLIB und der STEPLIB DD-Karte.

```
//TEST     JOB 6245,SCHMITT
//JOBLIB   DD  DSN=COMPLIB,DISP=SHR
//         DD  DSN=PRINTLIB,DISP=SHR
//STEP1    EXEC PGM=EINS
//STEPLIB  DD  DSN=LIB1,DISP=SHR
```

LIB1 und SYS1.LINKLIB werden in dieser Reihenfolge nach einem Programm EINS durchsucht.

```
//STEP2    EXEC PGM=ZWEI
```

COMPLIB, PRINTLIB und SYS1.LINKLIB werden in dieser Reihenfolge nach einem Programm ZWEI durchsucht.

```
//STEP3    EXEC PGM=DREI
//STEPLIB  DD  DSN=SYS1.LINKLIB,DISP=SHR
```

SYS1.LINKLIB allein wird nach einem Programm DREI durchsucht.

```
//STEP4    EXEC PL1CLG
//PL1.STEPLIB DD  DSN=LIB2,DISP=SHR
//           DD  DSN=LIB3,DISP=SHR,UNIT=DISK,VOL=SER=DA0009
```

PL1CLG ist ein katalogisiertes Verfahren. LIB2, LIB3 und SYS1.LINKLIB werden in dieser Reihenfolge nach dem im PL1-Step des Verfahrens verlangten Programm durchsucht. COMPLIB, PRINTLIB und SYS1.LINKLIB werden in dieser Reihenfolge nach den in den verbleibenden Jobsteps des Verfahrens verlangten Programmen durchsucht.

6.2.5 Programme in temporären Bibliotheken

Oft ist es so, daß die Ausgabe eines Jobsteps das im nachfolgenden Step auszuführende Programm wird. Die Linkage Editor-Ausgabe wird in der Regel im folgenden Step ausgeführt. Wird die Lademodul-Ausgabe des Linkage Editor nur für die Dauer des Jobs gebraucht, so wird sie in einer temporären Bibliothek gespeichert. Statt das auszuführende Programm jetzt mit einem Namen zu versehen ist es leichter, wenn man sich auf die DD-Karte rückbezieht, die die Datei beschreibt, in welcher das Programm enthalten ist. Der Step, welcher das Programm ausführt, muß nicht unmittelbar auf den das Programm erzeugenden Step folgen.

```
//LK       EXEC PGM=LINKEDIT
//LKEDOUT  DD  DSN=&&TEMP(GO),DISP=(NEW,PASS),UNIT=SYSDA,
//             SPACE=(1024,(200,20,1))
//STEP2    EXEC PGM=IEA001
//STEP3    EXEC PGM=*.LK.LKEDOUT
```

Falls ein katalogisiertes Verfahren aufgerufen wird, muß der Rückbezug den Stepnamen des katalogisierten Verfahrens einschließen. Wäre der oben angegebene Step LK Teil des katalogisierten Verfahrens LKED, so müßte wie folgt kodiert werden:

```
//STEP1     EXEC LKED
//STEP2     EXEC PGM=IEA001
//STEP3     EXEC PGM=*.STEP1.LK.LKEDOUT
```

Die Möglichkeit des Rückbezugs ist nicht allein auf temporäre Dateien beschränkt. Ein Programm kann mit Hilfe des Rückbezugs aus jeder beliebigen Datei ausgeführt werden.

```
//STEP1     EXEC PGM=IEFBR14
//PROGRAMM DD  DSN=COMPLIB(LAUF6),DISP=SHR
//STEP2     EXEC PGM=*.STEP1.PROGRAMM
```

STEP2 führt das Programm LAUF6 aus, das in der katalogisierten Datei COMPLIB enthalten ist. Das Programm IEFBR14 führt keine Funktionen aus, es ist sozusagen ein "Null"-Programm, das in der SYS1.LINKLIB enthalten ist. Das Programm führt nur eine Programm-End-Anweisung[1]) aus und dient als Rahmenprogramm für DD-Karten. Wie wir in späteren Kapiteln sehen werden, möchte man vielleicht einer Datei Platz auf einem Speicher mit direktem Zugriff zuordnen oder etwa eine Datei mit der Disposition DELETE versehen, ohne daß man ein großes Programm ausführt. Wie das obige Beispiel zeigt, könnte es auch sein, daß man eine Datei für einen späteren Rückbezug definieren muß. In all diesen Fällen nimmt man das "Null"-Programm IEFBR14. (Natürlich könnte LAUF6 auch mit einer JOBLIB oder STEPLIB DD-Karte ausgeführt werden.)

6.3 Der Name eines katalogisierten Verfahrens: prozedur

// *stepname* EXEC *prozedur* ruft ein auszuführendes katalogisiertes Verfahren auf. Kapitel 9 wird erklären, wie katalogisierte Verfahren erzeugt werden.

---

[1]) Anmerkung des Übersetzers: IEFBR14 ist folgendes Assembler-Programm.

```
IEFBR14  CSECT
         SR     15,15    LOESCHEN REG 15
         BR     14       RUECKSPRUNG ZUR ADR. IN REG 14
         END
```

```
//STEP1    EXEC PL1CLG
//STEPA    EXEC FORTCL
//         EXEC COBOLC
```

Wie das dritte Beispiel für den Aufruf eines katalogisierten Verfahrens zeigt, kann der Stepname in der aufrufenden EXEC-Karte auch weggelassen werden. Der Aufruf besteht aus dem Prozedurnamen allein oder aus dem Schlüsselwort PROC= *prozedur*:

```
//         EXEC PL1CLG
```

UND

```
//         EXEC PROC=PL1CLG
```

sind äquivalente Aufrufe für PL1CLG.

6.4 Schlüsselwortparameter

Die Schlüsselwortparameter (PARM, TIME usw.) werden auf der EXEC-Karte nach dem Programm- bzw. Verfahrensnamen kodiert. Sie gelten nur für den gerade ausgeführten Step.

```
//STEP1    EXEC PGM=STUDY,TIME=4,PARM=LIST
```

Die Parameter TIME und PARM gelten nur für STEP1. Die Schlüsselwortparameter können in beliebiger Reihenfolge nach dem Parameter PGM bzw. nach dem Verfahrensnamen kodiert werden.

Die Schlüsselwortparameter können zu jedem beliebigen Step eines katalogisierten Verfahrens hinzugefügt werden, indem man den Stepnamen an das Schlüsselwort anhängt.

```
//         EXEC PL1CLG,TIME.GO=4
```

Der Parameter TIME wird dem Step GO des Verfahrens PL1CLG hinzugefügt. Jeder im Step GO des Verfahrens bereits kodierter Parameter TIME wird dadurch überschrieben. Andere Parameter in diesem oder anderen Steps des Verfahrens werden nicht beeinflußt.

Jeder Parameter kann sooft kodiert werden, wie das Verfahren Steps enthält. Jedoch müssen die Parameter auf der EXEC-Karte in der Reihenfolge kodiert werden, in der die Steps im Verfahren auftreten.

Heißen die Steps des Verfahrens PL1CLG, PL1, LKED und GO, so muß wie folgt kodiert werden:

```
//          EXEC PL1CLG,PARM.PL1=DECK,TIME.PL1=G,
//               TIME.LKED=7,TIME.GO=4
```

Man darf nicht wie im folgenden Beispiel kodieren, denn der LKED-Step geht dem GO-Step ja voraus.

```
//          EXEC PL1CLG,TIME.GO=24,TIME.LKED=7
```

Ein Parameter gilt für alle Steps des Verfahrens, wenn man den Stepnemen wegläßt[2]). Er kann vor oder nach Parametern kodiert werden, denen ein Stepname angehängt ist. Um die Wirkung eines Parameters in einem Step eines katalogisierten Verfahrens zu unterdrücken, kodiert man nur das Schlüsselwort mit dem angehängten Stepnamen, gefolgt von einem Gleichheitszeichen.

```
//          EXEC PL1CLG,TIME.PL1=,PARM.PL1=MAP,
//               PARM.LKED=DECK,ACCT=6
```

Der TIME-Parameter im PL1-Step wird unterdrückt, der Parameter ACCT bezieht sich auf alle Steps, und die beiden PARM-Parameter gelten für die Steps PL1 und LKED.

6.5 Parameterübergabe an Jobstep: PARM

Der Parameter PARM=*wert* übergibt dem Jobstep zur Zeit der Step-Initialisierung Steuerinformationen. Die Angabe Wert kann 1 bis 100 Zeichen lang sein.

```
//STEP1     EXEC PGM=EINS,PARM=XREF
```

Wenn die Angabe Wert aus mehreren, durch Kommas getrennten Unterparametern besteht, oder wenn sie Sonderzeichen (Leerstelle ,./'()* &+-=) enthält, muß sie in Hochkommas eingeschlossen werden. Ein Hochkomma in der Angabe Wert muß mit zwei aufeinanderfolgenden Hochkommas kodiert werden:

---

[2]) Das bedeutet im einzelnen, daß beispielsweise der PARM-Parameter für den ersten Step gilt, während die PARM-Parameter in den nachfolgenden Steps des Verfahrens dadurch in ihrer Wirkung unterdrückt werden.

```
PARM='O''CLOCK,XREF,SIZE=100'
```

Auch Klammern können benutzt werden, um mehrere Unterparameter einzuschließen; dann müssen aber Sonderzeichen enthaltende Unterparameter zusätzlich in Hochkommas eingeschlossen werden:

```
PARM=('O''CLOCK',XREF,'SIZE=100')
```

Das Zeichen & muß doppelt kodiert werden, es sei denn, daß es einen symbolischen Parameter bezeichnet. Die Angabe Wert darf nur auf einer Folgekarte fortgesetzt werden, wenn sie in Klammern eingeschlossen wird, und wenn die Unterbrechung nach einem vollständigen Unterparameter, das Komma mit eingerechnet, erfolgt.

```
//STEP1    EXEC PGM=EINS,PARM=('K=6',
//               FUENF,'I=3','J=4',SIEBEN)
```

Die Werte K=6, FUENF, I=3, J=4 und SIEBEN werden durch den Parameter PARM dem Step übermittelt.

```
//         EXEC PL1CLG,PARM=STOP
```

Die Angabe STOP wird dem ersten Step des Verfahrens übergeben, alle PARM-Werte in den folgenden Steps des Verfahrens werden in ihrer Wirkung unterdrückt.

```
//         EXEC PL1CLG,PARM.LKED=MAP,PARM.GO=LIST
```

Der Wert MAP wird dem Step LKED, und LIST dem Step GO des Verfahrens übergeben.

Man muß wissen, welche Angaben das Verarbeitungsprogramm zur Übergabe erwartet. Wird PARM weggelassen, so werden keine Werte übermittelt. Von IBM zur Verfügung gestellte Programme, wie Compiler und der Linkage Editor erwarten Angaben für die verschiedenen Ausführungsoptionen. Programme, die im Assembler geschrieben werden, können ebenfalls PARM-Werte annehmen. Wenn das System einem Programm die Kontrolle übergibt, adressiert Register 1 ein Vollwort, das seinerseits einen Speicherbereich adressiert. Das erste Halbwort dieses Bereiches enthält die Anzahl der Zeichen in der Angabe Wert, der Rest des Bereiches enthält die Angabe Wert selbst.

Bei den IBM-Programmen existieren drei Stufen für die Angabe Wert, nämlich:

- Bei der Systemgenerierung unterstellte Werte,
- die Angaben für PARM in den katalogisierten Verfahren, wel-

che die Unterstellungen überschreiben, und
- PARM-Werte, welche in den EXEC-Karten zum Aufruf der Verfahren kodiert werden.

Wird ein einzelner PARM-Wert in einem Step eines katalogisierten Verfahrens überschrieben, so werden alle PARM-Unterparameter in diesem Verfahrensstep unterdrückt, und die unterstellten Werte werden wieder wirksam. Z.B. könnte ein FORTRAN Compile-Verfahren folgende Angaben enthalten:

```
//FORTC    PROC
//FORT     EXEC PGM=IEYFORT,PARM='LIST,MAP'
```

Wird die PARM-Angabe beim Verfahrensaufruf überschrieben, so werden alle PARM-Unterparameter überschrieben.

```
//         EXEC FORTC,PARM.FORT=DECK
```

Die Unterparameter LIST und MAP werden durch DECK ersetzt, und der Step FORT lautet bei der Ausführung:

```
//FORT     EXEC PGM=IEYFORT,PARM=DECK
```

Das Programm LOADER kann benutzt werden, um die Verarbeitung durch den Linkage Editor und die Programmausführung in einem einzigen Step zu vereinen. Die Parameter für das Programm LOADER und das auszuführende Programm müssen in einem einzigen PARM-Feld kodiert werden. Die Angaben für den LOADER müssen zuerst kodiert werden; durch einen Schrägstrich getrennt, folgen die Angaben für das Programm.

```
//LKEDGO   EXEC PGM=LOADER,PARM='MAP,PRINT/RUN'
```

MAP und PRINT werden dem Loader übergeben, RUN wird dem auszuführenden Programm übermittelt.

```
//LKEDGO   EXEC PGM=LOADER,PARM=MAP
```

MAP wird dem LOADER übergeben.

```
//LKEDGO   EXEC PGM=LOADER,PARM='/RUN'
```

RUN wird dem auszuführenden Programm übermittelt.

6.6 Abrechnungsinformationen für den Jobstep: ACCT

Abrechnungsinformationen für Jobsteps müssen von der Installation festgesetzt werden. Um solche Abrechnungsinformationen für den Step zur Verfügung zu stellen, kodiert man ACCT=(*abrechnungsinformationen*). Die Abrechnungsinformationen können aus einem oder mehreren, durch Kommas getrennten Unterparametern bestehen. Sie dürfen 142 Zeichen, die Kommas mitgerechnet, nicht überschreiten. Z.B. ACCT=(2645,30, 17). Die äußeren Klammern dürfen wegfallen, falls nur ein Parameter vorhanden ist.: ACCT=6225.

Falls die Unterparameter ein Sonderzeichen (Leerstelle, ./'()*%+=), den Bindestrich ausgenommen, enthalten, werden sie in Hochkommas eingeschlossen. Ein Hochkomma im Text wird wieder durch zwei Hochkommas kodiert: ACCT=(2645,'O''CLOCK','T=7').

```
//STEP1    EXEC PGM=EINS,ACCT=(24,53,
//               45,'A=17')
```

Die Abrechnungsinformationen dürfen auf einer Folgekarte fortgesetzt werden, wenn man sie nach einem vollständigen Unterparameter, das Komma mitgerechnet, unterbricht.

```
//         EXEC ALGOLCLG,ACCT=2647
```

Die Abrechnungsinformation gilt für jeden Step des Verfahrens ALGOLCLG.

```
//         EXEC PL1CLG,ACCT.PL1=('T=7','K=9'),
//              ACCT.GO=('T=9')
```

T=7 und K=9 werden dem Step PL1, und T=9 dem Step GO des Verfahrens PL1CLG zur Verfügung gestellt.

6.7 Dispatching-Priorität: DPRTY (nur MVT)

Man kodiert DPRTY=(*v1, v2*), um die Dispatching-Priorität zu setzen. *Dispatching-Priorität* ist die Priorität, mit der die Steps verschiedener Jobs im Speicher die Kontrolle des Rechners erhalten. In einem MVT-Multiprogramming-System erhält der Step mit der höchsten Priorität

die Kontrolle des Rechners, und zwar solange, bis er nur noch auf die Beendigung einer eingeleiteten E/A-Operation warten kann, bis sein Zeitintervall abläuft oder bis ein Step mit einer höheren Dispatching-Priorität den Rechner beansprucht. Die Kontrolle des Rechners wird dann dem darauf wartenden Step der höchsten Dispatching-Priorität übergeben.

Die dem Step zugewiesene Dispatching-Priorität wird errechnet als 16(v1)+v2, wobei v1 und v2 Werte von 0 bis 15 annehmen können. Der Wert 15 für v1 sollte für das System reserviert bleiben. Wird v1 nicht kodiert, so wird v1=0 angenommen, wird dagegen v2 nicht kodiert, so ist der angenommene Wert 11. DPRTY=(10,2) ergibt eine Dispatching-Priorität von 162, DPRTY=10 eine Priorität von 171 und DPRTY=(,2) eine solche von 2.

Wird DPRTY nicht kodiert, so unterstellt das System eine Dispatching-Priorität von 16 · (Jobauswahl-Priorität)+11. Die Jobauswahl-Priorität ist der durch den Parameter PRTY gesetzte Wert.

```
//STEP1    EXEC PGM=EINS,DPRTY=(5,2)
```

Die Dispatching-Priorität ist 82.

```
//         EXEC PL1CLG,DPRTY.LKED=(,2),DPRTY.GO=3
```

Die Dispatching-Priorität des Steps LKED des katalogisierten Verfahrens PL1CLG ist 2 und die des Steps GO 59. Falls v2 nicht kodiert wird, können die äußeren Klammern wegfallen.

```
//         EXEC PL1CLG,DPRTY=13
```

Die Dispatching-Priorität wird für jeden Step des katalogisierten Verfahrens PL1CLG auf 219 festgesetzt.

In Systemen mit der Einrichtung Time-Slicing werden die Jobs, die dieser Einrichtung unterzogen werden sollen, auf Grund ihrer Dispatching-Priorität ausgewählt. Ein Job wird Teil einer Gruppe von Jobs, die Time-Slicing erfahren sollen, indem man v1 als Gruppenpriorität kodiert, und entweder v2 wegläßt oder gleich 11 setzt.

6.8 Dumps bei abnormaler Beendigung: SYSUDUMP, SYSABEND

Es ist oft schwierig, die Fehlerquelle bei abnormaler Beendigung eines Programms zu finden. Die DD-Karte SYSUDUMP stellt einen hexadezimalen Speicherabzug (Dump) vom Programmbereich des Benutzers zur Verfügung. Dieser enthält die Inhalte der Register, eine Liste aufgerufener Unterprogramme und Benutzerdateien. Die DD-Karte SYSABEND stellt zusätzlich einen Speicherabzug des Nucleus des Systems zur Verfügung. Man sollte SYSUDUMP verwenden, es sei denn, man hat einen besonderen Grund, Speicherfelder des Nucleus untersuchen zu müssen. Falls man ihn wirklich benötigt, kann ein Speicherabzug äußerst wertvoll sein; er verschwendet aber nur Papier und Druckzeit, falls er nicht nötig ist.

Die Karte SYSUDUMP oder die Karte SYSABEND muß in jeden Step gelegt werden, für den ein Dump möglich sein soll. Ein Dump wird natürlich nur dann erzeugt, wenn der Step abnormal beendet.

```
//STEP1    EXEC PGM=EINS
//SYSUDUMP DD  SYSOUT=A
//STEP2    EXEC PGM=ZWEI
//SYSABEND DD  SYSOUT=A
```

Bei katalogisierten Verfahren muß der Name des Steps vorangestellt werden, für den der Dump gewünscht wird. Kapitel 9 bringt eine vollständige Beschreibung für das Hinzufügen von DD-Karten zu katalogisierten Verfahren.

```
//STEP1    EXEC PL1CLG
              .
              .
              .
//GO.SYSUDUMP DD  SYSOUT=A
```

Die Karte SYSUDUMP wird für den Step GO des Verfahrens PL1CLG wirksam.

SYSUDUMP und SYSABEND sind beides DD-Karten; ihre Parameter werden in späteren Kapiteln beschrieben. Obwohl auf dieser DD-Karte Parameter kodiert werden können, um den Dump auf eine Platte oder ein Magnetband zu legen, reicht jedoch die Angabe SYSOUT=A gewöhnlich aus, wenn der Dump nur gedruckt werden soll.

6.9 Zusammenfassung

Die EXEC-Karte benennt entweder das auszuführende Programm oder ruft das katalogisierte Verfahren auf. Die bisher behandelten Parameter der EXEC-Karte sind:

prozedur

PROC=*prozedur*

PGM=*programm*

PGM=*.*rückbezug*

// *stepname* EXEC ----------,PARM=*wert*

// ACCT=(*abrechnungsinformationen*),PRTY=(*v1, v2*)

Um Programme aus privaten Bibliotheken auszuführen, benötigt man JOBLIB und STEPLIB DD-Karten. PGM=IEFBR14 ruft das sogenannte "Null"-Programm auf. Eine SYSUDUMP oder SYSABEND Karte muß eingefügt werden, will man einen Dump für den Fall der abnormalen Stepbeendigung erhalten. Der folgende Job demonstriert EXEC-Karten.

```
//TEST1    JOB 5545,'PETER STEIN'
//JOBLIB   DD  DSN=SYS1.PROGRAMM,DISP=SHR
```

Es können jetzt private Programme ausgeführt werden, die in der Datei SYS1.PROGRAMM enthalten sind.

```
//EINS     EXEC PGM=PROGM1,PARM='SIZE=200'
```

Das Programm PROGM1 wird ausgeführt. Es ist entweder in der SYS1.LINKLIB oder in der SYS1.PROGRAMM enthalten. SIZE=200 wird diesem Programm als Parameter übergeben.

```
//ZWEI     EXEC PGM=PROGM2,DPRTY=(6,4)
//STEPLIB  DD  DSN=PRIVLIB,DISP=SHR
```

Das Programm PROGM2 wird ausgeführt. Es ist entweder in der SYS1.LINKLIB oder in der PRIVLIB enthalten. Die Dispatching-Priorität ist 100.

```
//DREI     EXEC PGM=IEFBR14
```

Step DREI führt das "Null"-Programm aus.

```
//VIER     EXEC ASMCLG,DPRTY=(,12),PARM.ASM=LIST,
//              ACCT.LKED=(3345,12),PARM.GO='VAR1=22,VAR2=24'
```

Es wird das katalogisierte Verfahren ASMCLG aufgerufen. Jeder Step im Verfahren bekommt die Dispatching-Priorität 12. Der PARM-Wert LIST wird dem Step ASM übergeben, der Step LKED erhält Abrechnungsinformationen (3345,12), und der Step GO bekommt die PARM-Werte VAR1=22 und VAR2=24.

```
//LKED.SYSABEND DD  SYSOUT=A
```

Sollte der LKED-Step abnormal enden, so wird ein Dump vom Programmbereich und vom Nucleus des Systems erzeugt.

```
//GO.SYSUDUMP DD  SYSOUT=A
```

Sollte der GO-Step abnormal beenden, so wird ein Dump des Programmbereiches erzeugt. Die katalogisierten Verfahren müssen in der Reihenfolge ihrer Steps überschrieben werden. Hier ist unterstellt, daß der GO-Step dem LKED-Step im Verfahren folgt.

7. Parameter der JOB- und EXEC-Karten

Verschiedene Parameter können auf der JOB- oder der EXEC-Karte kodiert werden, nämlich: Zeitbegrenzungen, Bedingungen für das Überspringen eines Jobsteps, Region Größe, Rollout/Rollin-Angaben und Restart-Angaben. Wird der betreffende Parameter auf der JOB-Karte kodiert, so bezieht er sich auf jeden Step des Jobs, und die entsprechenden Parameter auf den EXEC-Karten werden ignoriert. Das ist sehr bequem, wenn der Job viele Steps enthält. Der Parameter braucht nur einmal auf der JOB-Karte kodiert zu werden.

7.1 Zeitbegrenzung: TIME (nur MVT)

Der Parameter TIME setzt für den Rechner ein Zeitintervall, das für den ganzen Job gilt, wenn es in der JOB-Karte kodiert wird. TIME kann aber auch in der EXEC-Karte kodiert werden, um das Zeitintervall für den Rechner für nur einen bestimmten Step zu setzen. Man kodiert TIME=*minuten* oder TIME=(*minuten, sekunden*) in der JOB- oder EXEC-Karte. Die Minuten können Werte zwischen 0 und 1439 (24 Stunden) annehmen; der Wert sekunde muß kleiner als 60 sein. Überschreitet die Gesamtrechnerzeit des Jobs die in der JOB-Karte gesetzte Zeitbegrenzung, bzw. überschreitet die verstrichene Rechnerzeit des Steps die Zeitbegrenzung dieses Steps, so wird der ganze Job abnormal beendet.

Die Verwendung des TIME-Parameters ist praxisnah. Sie verhindert den Verlust von Maschinenzeit, falls das Programm in eine nicht endende Schleife geht. Der TIME-Parameter setzt die Zeitbegrenzung für die Rechnerzeit, nicht für die tatsächlich ablaufende Zeit. Die Rech-

nerzeit (CPU[1])time) ist die Zeit, in der für den Job Maschinenbefehle ausgeführt werden. Sie enthält weder die Wartezeit (Wait Time, das ist die Zeit, in der der Job auf die Beendigung der eingeleiteten E/A-Operationen wartet), noch die Systemzeit, noch die Zeit, in der andere Jobs während des Multiprogramming-Betriebes den Rechner benutzen.

Ein Job wird abnormal beendet, falls er für dreißig aufeinanderfolgende Minuten im Wartestatus ist. Um diese Begrenzung, sowie jede Rechnerzeitbegrenzung aufzuheben, kodiert man TIME=1440. Wird der TIME-Parameter nicht kodiert, so wird eine, in der katalogisierten Reader-Prozedur festgehaltene Zeitbegrenzung unterstellt.

Ist die Einrichtung System Management Facility installiert, so ist damit ein Mittel vorhanden, das normale 30-Minuten-Zeitintervall beliebig zu verändern. Zusätzlich kann die Installation eigene Vorkehrungen treffen, die Rechnerzeitbegrenzung für einen Job oder einen Step auszuweiten. Überschreitet ein Job seine Zeitbegrenzung, so wird ihm ein weiterer Zeitspielraum eingeräumt, damit er Gelegenheit hat, ordnungsgemäß zu enden.

7.1.1 Der Parameter TIME in der EXEC-Karte

```
//STEP1    EXEC PGM=EINS,TIME=(10,30)
```

Die Zeitbegrenzung für den Step wird auf 10 Minuten und 30 Sekunden festgesetzt. Der Step kann trotzdem eher beendet werden, nämlich dann, wenn das in der JOB-Karte gesetzte Zeitintervall überschritten wird.

```
//STEP2    EXEC PGM=EINS,TIME=7
```

Die Zeitbegrenzung wird auf 7 Minuten festgesetzt. Die äußeren Klammern fallen weg, wenn der Wert sekunden nicht kodiert wird.

```
//         EXEC PL1CLG,TIME.LKED=(,15),TIME.GO=1
```

Die Zeitbegrenzung für den LKED-Step wird auf 15 Sekunden und die für den GO-Step des katalogisierten Verfahrens PL1CLG auf 1 Minute festgesetzt.

```
//         EXEC PL1CLG,TIME=3
```

Die Zeitbegrenzung wird für jeden Step des katalogisierten Verfahrens PL1CLG auf 3 Minuten festgesetzt.

[1]) CPU=Central Processing Unit

7.1.2 Der Parameter TIME in der JOB-Karte

```
//TEST      JOB (54621,5,2),'ZEIT GRENZE',TIME=9
```

Der Job wird nach 9 Minuten Rechnerzeit abnormal beendet. Er wird eher abgebrochen, falls eine, in einer EXEC-Karte des Jobs gesetzte Zeitbegrenzung überschritten wird.

7.2 Bedingungen für das Überspringen von Jobsteps: COND

Jeder Jobstep kann, wenn er normal endet, dem System einen Rückkehr-Code übergeben. Der Parameter COND gestattet es, die Ausführung von Steps vom Rückkehr-Code vorangegangener Steps abhängig zu machen. Bricht beispielsweise eine Compilation ab, so besteht keine Notwendigkeit den nachfolgenden Linkage Editor-Step und den Ausführungsstep durchzuführen. Läßt man den COND-Parameter weg, so erfolgt kein Test, und die Ausführung der Steps geht normal vonstatten.

Programme in PL/I (durch den Aufruf der Routine IHESARC), in FORTRAN (durch die Anweisung STOP) und in der Assembler-Sprache (durch das Laden von Register 15 mit dem Rückkehr-Code) können einen Rückkehr-Code setzen. RPG-Programme setzen automatisch einen Rückkehr-Code. COBOL und ALGOL sind nicht in der Lage Rückkehr-Codes zu setzen.

Die Rückkehr-Codes variieren zwischen 0 und 4095. Die Rückkehr-Codes, die von den Compilern und vom Linkage Editor ausgegeben werden, lauten wie folgt:

- 0 Keine Fehler oder Warnungen entdeckt.
- 4 Warnungen (vielleicht Fehler) diagnostiziert. Die Ausführung könnte erfolgreich sein.
- 8 Ernsthafter Fehler wurde entdeckt, die Ausführung ist zweifelhaft.
- 12 Schwere Fehler, die Ausführung ist unmöglich.
- 16 Der Fehler ist so, daß Abbruch erfolgt.

Der COND-Parameter in der EXEC-Karte kann die Rückkehr-Codes eines oder aller vorausgegangener Steps testen. Falls irgendeiner der Tests positiv ausfällt, wird der testende Step übersprungen. Der Para-

meter COND in der EXEC-Karte erlaubt es aber auch, die Stepausführung von der abnormalen Beendigung vorausgegangener Steps abhängig zu machen. Wird der COND-Parameter kodiert, so werden, falls irgendein Test erfolgreich ist, alle nachfolgenden Steps übersprungen.

7.2.1 Der Parameter COND in der EXEC-Karte

Der Parameter wird in der EXEC-Karte kodiert als:
COND=((*zahl, operator, stepname*), ... , (*zahl, operator, stepname*))

Jede Zahl wird durch den Operator gegen den Rückkehr-Code verglichen. Es können bis zu acht solcher Tests kodiert werden. Wird der Stepname verwendet, so wird nur der Rückkehr-Code dieses Steps getestet. Wird der Stepname dagegen nicht kodiert, so wird der Test an den Rückkehr-Codes aller vorausgegangener Steps durchgeführt. Verläuft irgendeiner der Tests positiv, so wird der Step übersprungen. Die verwendeten Operatoren sind:

GT	(Greater Than)	Größer als
GE	(Greater than or Equal)	Größer als oder gleich
EQ	(Equal)	Gleich
LT	(Less Than)	Kleiner als
LE	(Less than or Equal)	Kleiner als oder gleich
NE	(Not Equal)	Ungleich

Z.B. bedeutet COND=(8,LT): Ist 8 kleiner als der Rückkehr-Code irgendeines vorausgegangenen Steps, so überspringe diesen Step.

```
//STEPA     EXEC PGM=EINS
//STEPB     EXEC PGM=ZWEI,COND=(4,EQ,STEPA)
```

STEPB wird übersprungen, falls 4 gleich dem von STEPA gesetzten Rückkehr-Code ist. Die äußeren Klammern fallen weg, falls nur ein Test kodiert wird.

```
//STEPC     EXEC PL1CLG,COND.GO=((12,GE),(8,EQ,STEP3),(4,EQ,STEPA)
```

Der GO-Step der PL1CLG-Prozedur wird übersprungen, falls 12 größer oder gleich dem Rückkehr-Code irgendeines vorausgegangenen Steps ist, falls 8 gleich dem Rückkehr-Code von STEPB ist, oder falls 4 gleich dem Rückkehr-Code von STEPA ist. Der COND-Parameter kann auf einer Folgekarte fortgesetzt werden, wenn die

Unterbrechung nach einem vollständigen Test, das Komma eingeschlossen, erfolgt.

```
//STEPD    EXEC PL1CLG,COND=(4,NE,STEPC.GO)
```

Jeder Step der PL1CLG-Prozedur wird übersprungen, falls 4 ungleich dem Rückkehr-Code von Step GO, des in STEPC aufgerufenen katalogisierten Verfahrens PL1CLG ist.

Findet sich im Job ein JCL-Fehler, so wird der Job in MFT- und MVT-Systemen nicht ausgeführt. Ein PCP-System wird den Job bis zu dem Step, der den Fehler enthält, ausführen. Dann erst wird der Job beendet. Wird ein Jobstep abnormal beendet, so kann kein Rückkehr-Code gesetzt werden, und alle nachfolgenden Steps werden übersprungen, es sei denn, die Parameter EVEN oder ONLY wurden kodiert.

COND=EVEN gestattet die Ausführung eines Jobsteps selbst dann, wenn vorausgegangene Steps abnormal enden. COND=ONLY bewirkt die Ausführung des Steps nur dann, wenn vorausgegangene Steps abnormal geendet haben. Dies wird oft benutzt, um Jobsteps zu überspringen. EVEN und ONLY können zusammen mit bis zu 7 Rückkehr-Code-Tests kodiert werden. Der Step mit dem Test COND=((EVEN,(8,EQ),(7,LT,STEPA)) wird übersprungen, falls 8 gleich dem Rückkehr-Code irgendeines vorausgegangenen Steps ist, oder falls 7 kleiner als der Rückkehr-Code von STEPA ist. Trifft das nicht zu, so wird dieser Step selbst dann ausgeführt (EVEN), wenn vorausgegangene Steps abnormal geendet haben.

Die Reihenfolge, in der die einzelnen Tests kodiert werden, ist ohne Belang. COND=((2,EQ),EVEN) und COND=(EVEN,(2,EQ)) sind gleichwertig.

EVEN und ONLY können in einem Step nicht gleichzeitig kodiert werden, denn sie schließen sich gegenseitig aus. Die Tests für EVEN und ONLY werden nur bei der Ausführung des Jobs durchgeführt. JCL-Fehler oder mangelnder Platz auf einem Plattenstapel bewirkt das Überspringen der restlichen Steps, unabhängig von den COND-Parametern.

```
//STEPA    EXEC PGM=EINS
//STEPB    EXEC PL1CLG,COND.LKED=EVEN,
//             COND.GO=((ONLY,(4,EQ,STEPB.LKED))
```

Der Step LKED des Verfahrens PL1CLG wird selbst dann (EVEN) ausgeführt, wenn vorausgegangene Steps abnormal enden. Der GO-Step wird übersprungen, falls 4 gleich dem Rückkehr-Code des

LKED-Steps ist, darüber hinaus wird er nur ausgeführt (ONLY), falls ein vorausgegangener Step abnormal geendet hat.

7.2.2 Der Parameter COND in der JOB-Karte

Der COND-Parameter in der JOB-Karte hebt alle COND-Parameter in EXEC-Karten des Jobs auf. Der Parameter wird kodiert als: COND=((*zahl, operator*),....,(*zahl, operator*)). Ein Stepname kann nicht verwendet werden, ebenso sind EVEN und ONLY nicht erlaubt. Die Tests werden am Ende eines jeden Jobsteps mit dem Rückkehr-Code des Steps durchgeführt. Fällt irgendeiner dieser Tests positiv aus, so werden die restlichen Steps übersprungen. Die Operatoren sind die gleichen wie in der EXEC-Karte; bis zu 8 Tests können kodiert werden.

```
//TEST     JOB 6245,SCHMITT,COND=((4,GT),(6,LT))
```

Der Job wird beendet, falls 4 größer als oder falls 6 kleiner als ein Rückkehr-Code ist. Das bedeutet, der Job wird beendet, es sei denn, der Rückkehr-Code ist 4, 5 oder 6.

```
//LAUF10   JOB (6452,200),JONES,COND=((4,EQ),(5,EQ),(6,EQ))
```

Der Job wird beendet, falls der Rückkehr-Code gleich 4, 5 oder 6 wird. Das ist gerade das Entgegengesetzte der obigen Bedingung.

7.3 Rollout/Rollin-Angaben: Roll (nur MVT)

Wurde die Einrichtung Rollout/Rollin in das System hineingeneriert, so ist es möglich, dem Jobstep, der seine Regiongröße überschreitet, zusätzlichen Kernspeicher zuzuordnen. Zuerst wird der Versuch unternommen, nicht vergebenen Hauptspeicher zu erhalten. Falls kein Platz verfügbar ist, wird er dadurch beschafft, daß ein anderer Job aus dem Speicher *rausgerollt* wird (darunter versteht man ein temporäres Auslagern des anderen Jobs in einen Hilfsspeicher). Der ausgelagerte Job muß mittels des ROLL-Parameters sein Einverständnis erklärt haben. Er kommt in den Speicher zurück, sobald der Jobstep, der ihn auslagerte, geendet ist. Falls kein Step dem Ausgelagertwerden zugestimmt hat, wird der Job mit dem Platzbedarf in eine Rollout Queue (Warteschlange) eingereiht. Er muß solange warten, bis ein anderer Job zustimmt oder bis Hauptspeicherplatz frei wird.

Wenn Rollout/Rollin wirksam werden soll, muß es sorgfältig geplant werden. Jeder möchte nur zu leicht andere Jobs auslagern, ohne daß er sich selbst bereit erklärt, ausgelagert zu werden. Datenfernverarbeitungsjobs mit der Option Autopoll dürfen nicht ausgelagert werden.

Die Rollout/Rollin-Angaben werden kodiert als:
ROLL=(*auslagerbar, lagert aus*). Man kodiert YES oder NO an den entsprechenden Stellen im ROLL-Parameter, wenn der Job auslagerbar ist bzw. selbst eine Auslagerung bewirkt. Z.B. bedeutet ROLL=(YES,NO), daß dieser Step ausgelagert werden kann, selbst aber keine Auslagerung verursacht.

In der katalogisierten Reader-Prozedur befindet sich für den ROLL-Parameter ein Standardwert, der unterstellt wird, wenn der ROLL-Parameter nicht kodiert wurde. Der Standardwert in der von der IBM zur Verfügung gestellten Reader-Prozedur lautet ROLL=(YES,NO).

7.3.1 Der Parameter ROLL in der EXEC-Karte

```
//STEP12   EXEC PGM=PROGM7,ROLL=(YES,YES)
```

STEP12 lagert aus, kann aber auch selbst ausgelagert werden.

```
//        EXEC PL1CLG,ROLL.LKED=(NO,NO),ROLL.GO=(YES,NO)
```

Der Step LKED im Verfahren PL1CLG ist weder auslagerbar noch lagert er aus. Der Step GO dagegen ist auslagerbar, lagert aber selbst nicht aus.

```
//        EXEC PL1CLG,ROLL=(NO,YES)
```

Keiner der Steps des Verfahrens PL1CLG läßt sich auslagern, dagegen werden alle Steps bei Bedarf eine Auslagerung verursachen.

7.3.2 Der Parameter ROLL in der JOB-Karte

Der ROLL-Parameter in der JOB-Karte bezieht sich auf alle Steps des Jobs. Irgendwelche ROLL-Parameter in den EXEC-Karten werden dann ignoriert.

```
//GROSS    JOB 5426,'O''HENRY',ROLL=(NO,NO)
```

Kein Step dieses Jobs versucht eine Auslagerung, noch läßt er sich auslagern.

7.4 Regiongröße: REGION (nur MVT)

OS-MVT-Systeme ordnen den Programmen, entsprechend ihrer Anforderung, zusammenhängenen Kernspeicher zu. Der Parameter REGION in der EXEC-Karte fordert den Speicher für den betreffenden Jobstep an. REGION, in der JOB-Karte kodiert, ordnet allen Steps die gleiche Regiongröße zu. Läßt man den REGION-Parameter weg, so wird ein in der katalogisierten Reader-Prozedur angegebener Wert unterstellt.

Mit dem REGION-Parameter wird ebenso Speicher einer zweiten Hierarchie-Stufe angefordert. Die Kernspeichereinheit IBM 2361 gestattet bis zu 8 Millionen Bytes Großraumspeicher an den Rechner anzuschließen. Der Großraumspeicher ist relativ langsam (2 bis 4 mal langsamer als ein Modell-50-Kernspeicher), jedoch sehr billig. Ist eine IBM 2361 Kernspeichereinheit installiert, so wird sie als Hierarchie-Stufe 1 angesprochen. Der Hauptspeicher hat dann die Hierarchie-Stufe 0.

Es wird REGION= *haupt*K für Hauptspeicher, bzw. REGION=(*haupt*K, *hierarchie*K) für Haupt- und Hierarchie-Speicher kodiert. Haupt und hierarchie geben die Anzahl von 1024-Byte Blöcken an, die für Hierarchie-Stufe 0 und Hierarchie-Stufe 1 zugeordnet werden sollen. Die im Kapitel 13 beschriebenen Linkage Editor-Steuer-Karten zeigen die Zuordnung von Programmen zu den Hierarchie-Stufen 0 und 1.

REGION=104K	Es werden 104K Bytes Hauptspeicher zugeordnet.
REGION=(,104K)	Es werden 104K Bytes Speicher der Hierarchie-Stufe 1 zugeordnet.
REGION=(104K,228K)	Es werden 104K Bytes Hauptspeicher und 228K Bytes Speicher der Hierarchie-Stufe 1 zugeordnet.

Die Werte haupt und hierarchie können bis zu 5 Ziffern groß sein. Jedoch dürfen weder der Wert haupt noch der Wert haupt zusammen mit dem Wert hierarchie 16 383 überschreiten (bzw. den maximal am Rechner verfügbaren Speicher). Bei einem einzigen Modell 1 2361 darf hierarchie 1024 und bei einem einzigen Modell 2 darf hierarchie 2048 nicht überschreiten. Die Werte haupt und hierarchie sollten gerade Zahlen sein; falls sie ungerade sind, werden sie vom System aufgerundet. Wird Hierarchie-1-Speicher verlangt, ohne daß er an den Rechner angeschlossen ist, so wird - falls vorhanden - Hierarchie-0-Speicher zu-

geordnet. (Es kann sein, daß der zweite Hierarchie-0-Bereich nicht dicht an den Bereich der Region anschließt.)

Wenn der Parameter REGION nicht kodiert wird, und daher eine Region von der in der Reader-Prozedur unterstellten Größe zugeordnet wird, so kann das nur im Hierarchie-0-Speicher geschehen. Das System beschafft sich immer eine Hierarchie-0-Region, die groß genug ist, um den Initiator aufzunehmen (falls der Terminator in der Link Pack Area resident ist, sind das 12K). Das geschieht sogar, wenn der REGION-Parameter keinen Hierarchie-0-Speicher anfordert. Da die Programme nur vom Linkage Editor in den Hierarchie-1-Speicher dirigiert werden können, sind zwei Kopien von einem Programm erforderlich, wenn es in jeder der beiden Hierarchien ausführbar sein soll.

Ist Hierarchie-Speicher an den Rechner angeschlossen, wurde aber keine Hierarchie-Unterstützung generiert, so wird der Hierarchie-Speicher als Erweiterung des Hauptspeichers behandelt. Es läßt sich nicht leicht kontrollieren, wo ein bestimmtes Programm ausgeführt wird; caveat emptor.

Das Abschätzen der benötigten Regiongröße ist schwierig. Verlangt man zu wenig, so wird der Step abnormal beendet. Wird zu viel verlangt, so drückt der verschwendete Speicher die Leistungsfähigkeit des Multiprogramming-Systems herab. Die Load Liste des Linkage Editor gibt die Programmgröße an, der Benutzer muß zu ihr aber eine Schätzung des erforderlichen dynamischen Hauptspeichers hinzufügen. Der Job wird während der Ausführung abbrechen, falls er keinen dynamischen Speicher bekommen kann.

Zum Beispiel könnte ein FORTRAN-Job, der einige Berechnungen anstellt und danach die Ergebnisse druckt, ganz schön in seine Region passen. Da sich FORTRAN beim ersten Lesen oder Schreiben dynamischen Speicher für Puffer beschafft, könnte das Programm abnormal beenden, bevor noch die Ergebnisse gedruckt sind. Es gibt zahlreiche Wege diese Schwierigkeit zu umgehen (Checkpoint/Restart, Rollout/Rollin, anfängliches Eröffnen aller Dateien), die einfachste Lösung ist ein ausreichender Sicherheitszuschlag beim Abschätzen der Regiongröße.

7.4.1 Der Parameter REGION in der EXEC-Karte

Der Parameter REGION in der EXEC-Karte verlangt für die einzelnen Steps Speicher.

```
//STEPA    EXEC PGM=EINS,REGION=64K
```

STEPA werden 64K Bytes des Hauptspeichers zugeordnet.

```
//        EXEC PL1CLG,REGION.LKED=88K,REGION.GO=(20K,512K)
```

Dem LKED-Step des Verfahrens PL1CLG werden 88K Hauptspeicher zugeordnet. Dem GO-Step werden 20K Hauptspeicher und 512K Hierarchie-1-Speicher zugeordnet.

```
//        EXEC COBOLCLG,REGION=104K
```

Jedem Step des Verfahrens COBOLCLG werden 104K Bytes Hauptspeicher zugeordnet.

7.4.2 Der Parameter REGION in der JOB-Karte

Der REGION-Parameter in der JOB-Karte muß auch für die von irgendeinem Step geforderte maximale Region ausreichend Platz zur Verfügung stellen. Für die kleineren Steps bedeutet dies Kernspeicherverschwendung. Z.B. möge ein FORTRAN-H-Job 228K für die Übersetzung, 104K für den Linkage Editor und 52K für die Ausführung benötigen. Der REGION-Parameter in der JOB-Karte muß für den größten Step (228K) ausreichend Platz zur Verfügung stellen; er verschwendet dann 124K während des Linkage Editor-Laufes und 176K während des Go-Laufes. Da Kernspeicher in MVT-Systemen rar ist, sollte man die Regiongrösse wenn möglich in der EXEC-Karte angeben.

```
//JOB27    JOB 4625,SCHMITT,REGION=300K
```

Allen Steps in diesem Job werden 300K Bytes zugeordnet. REGION-Parameter in den EXEC-Karten des Jobs werden ignoriert.

7.5 Wiederanlaufangaben: RD, RESTART

Das System/360 erlaubt abnormal beendeten Programmen wieder anzulaufen, ohne daß der ganze Job noch einmal gefahren werden muß. Das Wiederanlaufen kann automatisch (das System läßt den Job sofort wieder anfahren) oder verzögert (deferred) erfolgen. Die letztere Möglichkeit gestattet es, vor der Wiedereingabe des Jobs seine Ausgabe zu untersuchen und eventuell Änderungen vorzunehmen.

Alle Programme mit längeren Laufzeiten sollten diese Wiederanlaufeinrichtung benutzen. Ein Hardware-, ein Programm-, ein System- oder ein Bediener-Fehler kann den Abbruch des Jobs bewirken. Es läßt sich beträchtliche Zeit sparen, wenn der Job statt am Anfang wieder an der Abbruchstelle anlaufen kann. Der Parameter RD (Restart Definition) steuert das automatische Wiederanlaufen, während der Parameter RESTART es dem Benutzer gestattet, den Job für ein verzögertes Wiederanlaufen abzugeben.

Das Wiederanlaufen kann am Jobstepanfang ohne extra Programmierung geschehen. Um mitten im Step wieder anlaufen zu können, muß man für sein Programm während der Stepausführung einen *Checkpoint* schreiben lassen, um von diesem Checkpoint wieder starten zu können. Während eines Jobs können mehrere Checkpoints geschrieben werden. Das geschieht durch die Ausführung des CHKPT-Makrobefehls der Assembler-Sprache. In COBOL gibt die RERUN-Anweisung das CHKPT-Makro aus. Ist das Programm nicht in COBOL oder Assembler kodiert, so muß ein Assembler-Unterprogramm geschrieben werden, das dann das CHKPT-Makro ausführt. Sobald der Checkpoint geschreiben wurde, wird, um ihn zu kennzeichnen, auf dem Konsol ein unverwechselbarer Name, die *Checkpoint-Identifikation*, angedruckt. Dieser Name wird vom Benutzer oder vom System bestimmt.

Das Wiederanlaufen kann automatisch vom letzten Checkpoint oder Jobstep eingeleitet werden. Der Wiederanlauf kann aber auch später in einem getrennten Lauf von jedem Jobstep oder Checkpoint aus erfolgen. Der automatische Wiederanlauf geschieht allerdings nur, wenn der Code der abnormalen Beendigung des Steps in einem Satz ausgewählter Codes enthalten ist. Dieser Satz wird bei der Systemgenerierung für die Installation festgelegt. Darüber hinaus erfolgt der automatische Wiederanlauf nur, wenn der Operator seine Zustimmung gibt.

7.5.1 Die Wiederanlaufangabe RD

Der RD-Parameter steuert das automatische Wiederanlaufen. Er kann außerdem die Wirkung des CHKPT-Makros unterdrücken, so daß keine Checkpoints geschrieben werden. Der Parameter wird kodiert als RD=*wiederanlaufbedingung*. Die folgende Tabelle gibt die Werte für die Wiederanlaufbedingung an.

Wiederanlaufbedingung	Automatischer Wiederanlauf?	CHKPT Unterdrückung?
R - Restart	Ja	Nein
NC - No Checkpoint	Nein	Ja
NR - No Automatic Restart	Nein	Nein
RNC - Restart and No Checkpoint	Ja	Ja

Beispielsweise gestattet RD=RNC das automatische Wiederanlaufen, unterdrückt aber das CHKPT-Makro. Falls automatisches Wiederanlaufen gefordert wird, geschieht es vom letzten Checkpoint des Steps (*Checkpoint Restart*) aus. Wurde dagegen kein Checkpoint geschrieben, weil das CHKPT-Makro unterdrückt worden war, weil es fehlte oder weil der Step abnormal endete, bevor das Makro ausgeführt worden war, so geschieht das Wiederanlaufen am Stepanfang (*Step Restart*). Wurde der RD-Parameter weggelassen, so geschieht der automatische Wiederanlauf nur, falls das CHKPT-Makro ausgeführt worden ist.

Der automatische Wiederanlauf verlangt vom System für die Dateien besondere Dispositionen. Läuft ein Step automatisch wieder an, so werden alle im Step erzeugten Dateien wieder gelöscht, alle bei der ersten Initialisierung des Steps bestehenden Dateien werden festgehalten. Geschieht der Wiederanlauf von einem Checkpoint, so werden alle gerade benutzten Dateien festgehalten. Jede Zeitbegrenzung für einen Step wird beim Wiederanlauf auf ihren Ausgangswert zurückgesetzt.

Auf der JOB-Karte in MFT- und MVT-Systemen darf für den Wiederanlauf MSGLEVEL=(0,0) weder kodiert noch unterstellt werden. Jeder Step im Job muß einen Stepnamen haben. Dieser muß im Job unverwechselbar sein, denn das System benutzt ihn für die Entscheidung, welcher Step wiederanlaufen soll.

7.5.2 Der Parameter RD in der EXEC-Karte

```
//JOB7      JOB 5426,SCHMITT,MSGLEVEL=1,...
```

MSGLEVEL=1 ist in der JOB-Karte bei MFT- und MVT-Systemen erforderlich.

```
//STEP1     EXEC PGM=CSMP,RD=R
```

STEP1 kann automatisch wiederanlaufen.

```
//          EXEC FORTCLG,RD.LKED=NR,RD.GO=NC
```

Der LKED-Step des Verfahrens FORTCLG läuft nicht automatisch wieder an. Der GO-Step läuft ebenfalls nicht automatisch wieder an, außerdem werden die CHKPT-Makros in ihrer Wirkung unterdrückt. Jeder Step des Verfahrens FORTCLG muß einen Stepnamen haben, sie müssen im Job unverwechselbar sein.

```
//          EXEC FORTCLG,RD=RNC
```

Für jeden Step des Verfahrens FORTCLG wird das automatische Wiederanlaufen zugelassen. Die Ausführung des Makros CHKPT dagegen wird unterdrückt.

7.5.3 Der Parameter RD in der JOB-Karte

Der in der JOB-Karte kodierte RD-Parameter bezieht sich auf jeden Step im Job. RD-Parameter in den EXEC-Karten des Jobs werden ignoriert.

```
//JOB8      JOB 4562,SCHMITT,RD=R,MSGLEVEL=1
```

Alle Steps innerhalb des Jobs können automatisch wiederanlaufen.

7.5.4 Abgabe eines Jobs für den Wiederanlauf: RESTART

Man kann die Ausgabe seines Jobs einer Prüfung unterziehen, und sich dann dafür entscheiden, ihn von einem Checkpoint oder vom Stepbeginn wiederanlaufen zu lassen. Ein verzögertes Wiederanlaufen des Jobs (Deferred Restart) kann unabhängig von der Art der Jobbeendigung - abnormal oder normal - und ohne Rücksicht darauf, ob ein automatischer Wiederanlauf geschah oder nicht, vonstatten gehen. Das verzögerte Wiederanlaufen erlaubt es dem Benutzer vor der Wiederabgabe des Jobs Daten zu verändern oder zu verbessern sowie Programmfehler zu beseitigen. Der in der JOB-Karte kodierte RESTART-Parameter verlangt vom System einen verzögerten Wiederanlauf.

7.5.5 Wiederanlauf am Stepbeginn

Ein verzögerter Wiederanlauf vom Jobstep veranlaßt das System, die Ausführung einfach mit einem bestimmten Step innerhalb des Jobs, unter

Umgehung aller davorliegenden Steps, zu beginnen. Von den vorausgegangenen Steps werden keine Dateien übergeben (PASS), denn diese Steps sind ja nicht ausgeführt worden. Rückbezüge können nicht verwendet werden, es sei denn im DCB-Parameter und im VOL-Parameter, der sich auf eine in einer DD-Karte angegebenen Datenträgernummer rückbezieht. Man beachte, daß in vorangegangenen Läufen erzeugte neue Dateien noch bestehen können. Der Versuch, sie noch einmal zu erzeugen, wird den Jobabbruch wegen "doppelter Datei auf dem gleichen Datenträger" zur Folge haben. Die JCL müßte modifiziert werden, um diese Beschränkungen zu umgehen.

Man untersuche stets die Ausgabe der vorausgegangenen Läufe, um die Bedingungen so herzustellen, daß der Step wieder anlaufen kann. Vielleicht ist es nicht möglich, den einen Step wieder anlaufen zu lassen. Falls eine, von einem vorausgehenden Step übergebene, temporäre Datei gelöscht worden ist, muß man mit dem Step wieder anfahren, der sie erzeugt hat. Falls eine übergebene Datei (DISP=(....,PASS)) festgehalten wurde (KEEP), muß man die Datei mit ihrem Namen, ihrer Einheit, ihrer Datenträgernummer und den Kennsatzinformationen beschreiben. Ist die Datei katalogisiert, so benötigt man nur ihren Namen und ihre Disposition. Wurden Dateien in vorangegangenen Steps auf Speichern mit direktem Zugriff erzeugt und festgehalten, so kann man entweder die Disposition in der DD-Karte von NEW auf OLD bzw. MOD ändern, oder man kann die Datei vor der Wiederabgabe des Jobs löschen. Es ist aber auch möglich, eine neue Datei anzulegen.

Der Wiederanlauf läßt sich durch das Katalogisieren der Dateien vereinfachen. Katalogisiert man sie bei ihrer Erzeugung, so kann man sich auf sie allein mit ihrem Namen beziehen, ohne daß die Einheit und die Datenträgernummer angegeben werden muß. Man kann den Notausgang der Datei-Disposition (das ist der dritte Parameter) benutzen, um neu erzeugte Dateien wieder zu löschen bzw. übergebene Dateien zu katalogisieren. Falls nötig, können natürlich noch zusätzliche Änderungen an der JCL vorgenommen werden. Man kann andere Einheiten oder Datenträger benutzen, Daten können modifiziert werden, und andere Dateien können benutzt werden.

Der Step-Wiederanlauf wird durch das Kodieren von RESTART=stepname in der JOB-Karte verlangt. Der Wert stepname gibt den Namen des Steps an, der wieder anlaufen soll.

```
//TEST      JOB 2256,SCHMITT,RESTART=STEP6
//STEP1     EXEC ...
                .
                .
                .
//STEP6     EXEC ...
```

Die Ausführung wird mit STEP6 wieder aufgenommen.

Der Wiederanlauf kann natürlich auch mit dem Step eines katalogisierten Verfahrens beginnen.

```
//TEST      JOB 2249,SCHMITT,RESTART=RUN.LKED
//STEP1     EXEC ...
//RUN       EXEC PL1CLG
```

Die Ausführung wird mit dem LKED-Step des Verfahrens PL1CLG wieder aufgenommen.

Wird RESTART=* in der JOB-Karte kodiert, so beginnt die Ausführung mit dem ersten Step. (Das Gleiche geschieht natürlich, wenn der RESTART-Parameter ganz fehlt.)

```
//TEST      JOB 2564,SCHMITT,RESTART=*
//GO        EXEC ...
```

Die Ausführung beginnt mit dem GO-Step.

7.5.6 Wiederanlauf von einem Checkpoint

Viele der für einen Step-Wiederanlauf erforderlichen Modifikationen fallen weg, wenn der Wiederanlauf von einem Checkpoint geschieht. Das System bewahrt die JCL des Jobs auf und verändert den Status neu erzeugter Dateien in OLD. Es bewahrt ebenso Einheiten und Datenträgerangaben für alle Dateien auf, außer für Dateien mit PASS. Man muß die JCL nur abändern, damit für übergebene Dateien die Informationen für Einheit, Datenträgernummer und Kennsatz zur Verfügung gestellt werden. Das ist nicht nötig, falls die übergebene Datei katalogisiert wurde. Andere vor der Wiederabgabe des Jobs mögliche JCL-Änderungen sind: Änderung vorhandener und Hinzufügung neuer DD-Anweisungen, Änderung von Daten und DUMMY-Setzung unnötiger DD-Karten etc.

Um einen Job an einem Checkpoint wieder anlaufen zu lassen, kodiert man RESTART=(*stepname, checkid*) in der JOB-Karte. Der Wert step-

name ist der Name des Steps, der wieder anlaufen soll; der Wert checkid ist der dem Checkpoint vom System oder vom Benutzer zugeordnete Name. RESTART=(*, checkid) läßt den Job im ersten Step wieder anlaufen. (Der Wert checkid muß in Hochkommas eingeschlossen werden, falls er Sonderzeichen (Leerstelle, ./' ()*&+-=) enthält; ein Hochkomma im Text wird durch zwei aufeinanderfolgende Hochkommas dargestellt.)

Geschieht der Wiederanlauf von einem Checkpoint, so muß eine SYSCHK DD-Karte, die die Checkpoint Datei beschreibt, hinzugefügt werden. Natürlich mußte vorher für das CHKPT Makro eine DD-Karte kodiert werden, die die Datei beschreibt, in die hinein der Checkpoint geschrieben wird. Die SYSCHK DD-Karte muß jetzt diese Datei beschreiben. Die SYSCHK DD-Karte wird unmittelbar hinter die JOB-Karte und hinter eine eventuell vorhandene JOBLIB Karte gelegt. Sie liegt damit vor der ersten EXEC-Karte.

```
//TEST     JOB 2566,JONES,RESTART=(STEP1.GO,CK10)
```

Der Wiederanlauf geschieht im GO-Step des von STEP1 aufgerufenen Verfahrens PL1CLG. Er wird dort vom Checkpoint CK10 vorgenommen.

```
//JOBLIB   DD  DSN=PROGLIB,DISP=SHR
//SYSCHK   DD  DSN=CHKPTLIB,DISP=OLD
```

Die SYSCHK DD-Karte folgt der JOB-Karte und der JOBLIB Karte.

```
//STEP1    EXEC PL1CLG
```

Die SYSCHK DD-Karte kann, wenn nötig, noch andere DD-Parameter enthalten. Ist die Checkpoint-Datei nicht katalogisiert, so müssen Angaben über Einheit und Datenträger zur Verfügung gestellt werden. Falls die Checkpoint-Datei eine untergliederte Datei ist, darf kein Gliedname kodiert werden. Die SYSCHK DD-Karte muß eine Disposition von OLD oder KEEP angeben oder implizieren.

7.6 Zusammenfassung

Die Parameter TIME, COND, ROLL, REGION und RD können in der JOB- oder EXEC-Karte kodiert werden. In der JOB-Karte kodierte Pa-

rameter überschreiben die entsprechenden in den EXEC-Karten des Jobs kodierten Parameter. Der Parameter RESTART wird für den verzögerten Wiederanlauf in der JOB-Karte kodiert.
Die vollständigen JOB-Karten-Parameter sind:

```
// jobname    JOB  (kostenstellennummer, abrechnungsinformationen),
//                  name, MSGLEVEL=(jcl, zuordnung),
//                  PRTY=priorität, MSGCLASS=klasse,
//                  CLASS=klasse, TYPRUN=HOLD,
//                  REGION=(hauptK, hierarchieK),
//                  TIME=(minuten, sekunden),
//                  COND=((zahl, operator), ...),
//                  RD=wiederanlaufbedingung,
//                  ROLL=(auslagerbar, lagert aus),
                              *
                             stepname
//                  RESTART=(--------, checkid)
```

Die vollständigen EXEC-Karten Parameter sind:

```
                    prozedur
                    PROC=prozedur
                    PGM=programm
                    PGM=rückbezug
// stepname EXEC    ----------, PARM=wert,
//                  ACCT=(abrechnungsinformationen),
//                  TIME=(minuten, sekunden),
//                  DPRTY=(v1, v2),
                          EVEN
                          ONLY
                         (zahl, operator, stepname)
//                  COND=(--------------------, .....),
//                  ROLL=(auslagerbar, lagert aus),
//                  REGION=(hauptK, hierarchieK), RD=wiederanlaufbedingung
```

8. Beschreibung der DD-Karte

Für jede im Step benutzte Datei muß nach der EXEC-Karte eine DD-Karte (Data Definition) eingefügt werden. Die DD-Karte gibt den Dateinamen, das E/A-Gerät, vielleicht einen bestimmten Datenträger und die Datei-Dispositionen an. Das System stellt sicher, daß dem Job die verlangten E/A-Geräte zugeordnet werden, bevor mit der Ausführung begonnen wird. Die pro Step maximal erlaubte Anzahl von DD-Karten ist 255.

Die DD-Karte kann dem System, über die genannten Angaben hinaus, Informationen über die Datei geben, wie: Organisationsform, Satzlänge, Blockung etc. Falls eine neue Datei auf einem Speicher mit direktem Zugriff eingerichtet wird, gibt die DD-Karte auch die zuzuordnende Speichermenge an. Ebenso kann die DD-Karte eine bestimmte Bandrolle einer aus mehreren Bandrollen bestehenden Banddatei benennen. Mit der DD-Karte kann der E/A-Vorgang optimalisiert werden, dadurch daß Dateien zusammen auf der gleichen Einheit angelegt werden oder dadurch daß sie voneinander getrennt auf verschiedene Einheiten gelegt werden. Die gleiche Trennung ist für E/A-Kanäle möglich.

Um eine Datei zu lesen oder zu schreiben, muß eine ganze Menge Informationen zur Verfügung gestellt werden: Die Organisationsform der Datei, der Satztyp, die Satzlänge, die Blocklänge, Angaben zur Prüfung, der Typ des E/A-Gerätes usw. Das System/360 bezieht diese Informationen aus drei Quellen:

- dem Data Control Block (DCB)
- der DD-Karte und
- dem Kennsatz (Data Set Label),

8.1 Data Control Block

Der Data Control Block ist eine Tabelle im Speicher, die die vom Programm benutzte Datei beschreibt. Er wird in der Regel vom Sprachübersetzer zur Verfügung gestellt. In der Assembler-Sprache kann er aber auch direkt kodiert werden. Der Sprachübersetzer füllt die verschiedenen Angaben für den Programmierer aus. Der Assembler-Programmierer muß seine eigenen Angaben zur Verfügung stellen; er hat dafür aber auch eine größere Kontrolle über die Datei.

Wenn die Datei eröffnet wird, werden die Angaben der DD-Karte in die leeren Felder des Data Control Block eingefügt. Anschließend wird der Dateikennsatz gelesen (falls es sich um eine existierende Datei mit einem Kennsatz handelt). Seine Informationen dienen dazu, die noch leeren Felder des Data Control Block aufzufüllen. Die Rangfolge ist folgende: Zuerst Data Control Block, dann DD-Karte und drittens Dateikennsatz.

Z.B. ist es nicht nötig, den Typ des Gerätes im Data Control Block zu kodieren, er wird in der DD-Karte zur Verfügung gestellt und macht damit das Programm vom Gerät unabhängig. Ebenso werden Satzlänge und Blockung oft im Data Control Block weggelassen, um später entweder in der DD-Karte oder im Dateikennsatz zur Verfügung gestellt zu werden.

Der Benutzer allerdings muß wissen, welche Felder im Data Control Block noch leer sind, damit er diese fehlenden Angaben später in der DD-Karte zur Verfügung stellen kann. Der Assembler-Programmierer wird das natürlich wissen, denn er kodiert ja seinen Data Control Block selbst. Die Benutzer anderer Programmiersprachen müssen diese Informationen im entsprechenden IBM-Programmer's Guide nachschlagen.

Da es mehrere Data Control Blocks geben kann, einen für jede Datei, wird jedem ein Data Definition Name (ddname) zugeordnet. Eine DD-Karte mit dem entsprechenden ddname muß für jede Datei eingefügt werden. Um weitere Informationen über die Datei zu erhalten, kann das System den Dateikennsatz lesen.

8.2 Format der DD-Karte

Das Format der DD-Karte ist wie folgt:

// ddname DD *wahlfreie Parameter*

Der ddname ist der Name der DD-Karte. Die wahlfreien Parameter bestehen aus drei Positionsparametern (DUMMY, DATA, und *) und mehreren Schlüsselwortparametern.

DUMMY	Gibt der Datei den Status einer Scheindatei.
DATA	Zeigt an, daß der DD-Karte unmittelbar Datenkarten mit // in den Spalten 1 und 2 folgen.
*	Zeigt an, daß der DD-Karte unmittelbar Datenkarten folgen.
DCB	Spezifiziert Parameter für den Data Control Block.
DSN	Gibt den Namen der Datei an.
UNIT	Benennt die E/A-Einheit.
DISP	Gibt die Datei-Disposition an.
VOL	Stellt Datenträgerinformationen zur Verfügung.
LABEL	Stellt Kennsatzangaben, sowie Dateischutzdaten zur Verfügung.
SPACE	Fordert Platz auf Speichern mit direktem Zugriff an.
SPLIT	Teilt den einzelnen Zylinder eines Speichers mit direktem Zugriff zwischen Dateien auf.
UCS	Gibt den Zeichensatz für den Druckvorgang an.
SYSOUT	Gibt eine Datei über den Ausgabestrom des Systems aus.
DDNAME	Stellt die Definition einer Datei zurück.
SEP	Separiert Dateien auf verschiedene Kanäle.
AFF	Legt Dateien auf den gleichen Kanal.
SUBALLOC	Vergibt den Platz einer Datei weiter, die auf einem Speicher mit direktem Zugriff liegt.

DUMMY, DATA und * sind Positionsparameter; sie müssen, wenn sie nötig sind, zuerst im Operandenfeld kodiert werden. (DUMMY, DATA und * schließen sich gegenseitig aus.) Die Schlüsselwortparameter können im Operandenfeld in beliebiger Reihenfolge kodiert werden; sie müssen aber nach den Positionsparametern stehen.

Die Parameter ddname, DCB, DSN, UNIT, DISP und VOL werden in diesem Kapitel beschrieben. Sie sind für viele DD-Karten und für eine Vielzahl von Geräten erforderlich. Die restlichen Parameter hängen

von dem betreffenden Gerät ab und werden nicht so oft benutzt. Sie werden in den folgenden Kapiteln beschrieben.

8.3 Data Definition Name: ddname

Der Wert ddname kann aus 1 bis 8 alphanumerischen (A-Z, 0-9) oder sogenannten nationalen (@ $ *) Zeichen bestehen. Jeder ddname in einem Step muß unverwechselbar sein. Falls in einem Step zwei gleiche ddname existieren, wird zwar die Geräte- und Platzordnung für jede Karte durchgeführt, doch werden alle Bezugnamen nur auf die erste Karte bezogen. Die Werte für ddname sind für die einzelnen Sprachübersetzer wie folgt.

Sprache	Datei angesprochen durch	erforderlicher ddname
ALGOL	ALGOL Einheit 0	//SYSIN DD ...
	ALGOL Einheit 1	//ALGLDD01 DD ...
	.	.
	.	.
	.	.
	ALGOL Einheit 15	//ALGLDD15 DD ...
Assembler	DCB=ddname	//ddname DD ...
COBOL	SELECT file ASSIGN TO ddname	//ddname DD ...
FORTRAN	FORTRAN Einheit 1	//FT01F001 DD ...
	.	.
	.	.
	.	.
	FORTRAN Einheit 99	//FT99F001 DD ...
PL/1	DECLARE ddname FILE oder OPEN FILE (ddname)	//ddname DD ...
RPG	filename	//filename DD ...

8.4 Parameterrückbezug: rückbezug

Der Parameterrückbezug wird in der DD-Karte benutzt, um Angaben von einer vorausgegangenen DD-Karte zu übernehmen. Er ist dem in Kapitel 6 gezeigten Rückbezug für die EXEC-Karte ähnlich: //STEP3 EXEC PGM=*.rückbezug. Die folgenden Beispiele zeigen den Rückbezug. Es werden Data Control Block-Angaben von einer vorausgegangenen DD-Karte kopiert.

```
//COMP      PROC
//STEP1     EXEC PGM=EINS
//DD1       DD  DCB=BLKSIZE=80,...
//DD2       DD  DCB=*.DD1,...
```

Um sich auf eine DD-Karte des gleichen Steps zu beziehen, kodiert man *.ddname.

```
//STEP2     EXEC PGM=ZWEI
//DD3       DD  DCB=*.STEP1.DD1,...
```

Um sich auf eine DD-Karte eines früheren Steps zu beziehen, kodiert man *.stepname.ddname.

Will man sich auf eine DD-Karte in einem katalogisierten Verfahren beziehen, so bezieht man sich zuerst auf den Stepnamen des die Prozedur rufenden Steps, dann auf den Stepnamen des Steps in der Prozedur und schließlich auf den Namen der DD-Karte. Ruft man das oben angegebene Verfahren COMP, so könnte folgendes kodiert werden.

```
//STEP3     EXEC COMP
//STEP4     EXEC PGM=VIER
//DD4       DD  DCB=*.STEP3.STEP1.DD1,...
```

8.5 Data Control Block-Angaben: DCB

Der Parameter DCB wird als DCB=unterparameter bzw. DCB=(unterparameter,..,unterparameter) kodiert. Der Anhang A führt alle DCB-Unterparameter auf, die möglich sind; einige der gebräuchlichsten DCB-Unterparameter werden hier beschrieben.

8.5.1 Dateiorganisation: DSORG

Zur Angabe der Organisationsform kodiert man DSORG=*organisation*. Die Organisationsform kann eine der folgenden sein: (Dateien sind unmovable, also unbeweglich, falls die Daten absolute Spuradressen enthalten.)

PS	Physical Sequential	Sequentiell
PSU	Physical Sequential Unmovable	Sequentiell (und unbeweglich)
PO	Partitioned Organization	Organisationsform der untergliederten Datei
POU	Partitioned Organization Unmovable	Organisationsform der untergliederten Datei (und unbeweglich)
IS	Indexed Sequential	Index Sequentiell
ISU	Indexed Sequential Unmovable	Index Sequentiell (und unbeweglich)
DA	Direct Access	Organisationsform für direkten Zugriff
DAU	Direct Access Unmovable	Organisationsform für direkten Zugriff (und unbeweglich)

```
//AUSGABE  DD  DCB=DSORG=PS,...
```

8.5.2 Länge des logischen Satzes: LRECL

Um die Länge eines logischen Satzes für feste und variable Sätze in Bytes anzugeben, kodiert man LRECL= *länge*. Bei Sätzen fester Länge ist LRECL der Satzlänge gleich. Bei Sätzen variabler Länge ist LRECL gleich der Länge des größten Satzes, die 4 die Satzlänge beschreibenden Bytes miteingeschlossen.

```
//AUSGABE  DD  DCB=(DSORG=PS,LRECL=80),...
```

8.5.3 Länge des Blockes: BLKSIZE

Um die Länge eines Blockes in Bytes anzugeben, kodiert man BLKSIZE= *blocklänge*.

Für Sätze fester Länge muß der Wert für BLKSIZE ein ganzzahliges Vielfaches des Wertes von LRECL sein. Für Sätze variabler Länge muß er gleich dem oder größer als der um 4 vermehrte LRECL-Wert sein (LRECL war die Länge des längsten Satzes).

```
//AUSGABE  DD  DCB=(DSORG=PS,LRECL=80,BLKSIZE=1600),...
```

DCB-Unterparameter können auf einer Folgekarte fortgesetzt werden, wenn nach einem vollständigen Unterparameter, das Komma eingeschlossen, abgebrochen wird.

8.5.4 Anzahl der Puffer: BUFNO

Zur Angabe der Pufferanzahl kodiert man BUFNO=*anzahl*.

```
//AUSGABE  DD  DCB=(DSORG=PS,LRECL=80,BLKSIZE=1600,BUFNO=2),...
```

8.5.5 Satzformat: RECFM

Um das Satzformat anzugeben, kodiert man RECFM=*format*. (siehe Abb. 8.1) Das Satzformat wird mit einem oder mehreren der folgenden Zeichen verschlüsselt.

U Sätze undefinierter Länge. LRECL sollte nicht kodiert werden. Der Wert für BLKSIZE muß gleich der Länge oder größer als die Länge des größten Satzes sein.

V Sätze variabler Länge. Falls die Sätze geblockt sind, muß der Wert für BLKSIZE gleich der um 4 vermehrten Länge des längsten Satzes sein.

F Sätze fester Länge. Für geblockte Sätze muß BLKSIZE ein Vielfaches von LRECL sein. (U, V und F schließen sich gegenseitig aus.)

B Die Sätze sind geblockt. (FB und VB sind zulässig; UB ist nicht zulässig.)

T Die Folgespureinrichtung (Track Overflow)wird benutzt. (siehe Kapitel 14)

S Standard-Blöcke (siehe Kapitel 14). Für Sätze mit dem Format F bedeutet das, es befinden sich keine verkürzten Blöcke oder nicht volle Spuren in der Datei. Für Sätze mit dem Format V heißt das, die logischen Sätze können sich über mehr als einen Block erstrekken.

A Ein ASA-Steuerzeichen ist das erste Byte eines jeden Satzes. (Steuerzeichen werden im Anhang B beschrieben.)

M Das erste Byte eines jeden Satzes ist ein Maschinensteuerzeichen. (A und M schließen sich gegenseitig aus.)

```
//AUSGABE   DD   DCB=(DSORG=PS,LRECL=80,BLKSIZE=1600,
//                BUFNO=2,RECFM=FB),...
//AUSGABE   DD   DCB=RECFM=FBSA,...
```

8.5.6 DCB-Angaben aus dem Kennsatz

Für Dateien mit Standard-Kennsätzen wird ein Kennsatz als Teil einer nicht temporären Datei erzeugt. Im Kennsatz werden die DCB-Angaben DSORG, RECFM, BLKSIZE und LRECL gespeichert. Sie werden so zu einem Teil der Datei. Wenn man sich auf eine bestehende Datei bezieht, müssen diese Angaben nicht mehr angegeben werden; das System beschafft sie sich aus dem Kennsatz der Datei. Falls alle erforderlichen DCB-Angaben im Data Control Block oder im Kennsatz enthalten sind, kann der DCB-Parameter wegfallen.

8.5.7 Übernahme von DCB-Angaben aus einer vorangegangenen DD-Karte oder Datei: rückbezug, name

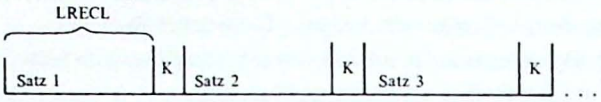
Es kommt oft vor, daß man eine Datei erzeugen möchte, die die gleichen DCB-Parameter hat, wie eine bestehende Datei. Man erreicht das, indem man entweder die andere Datei nennt oder indem man sich auf eine DD-Karte rückbezieht, die die gewünschten DCB-Angaben enthält.

DCB=(*name, unterparameter, . . . , unterparameter*)

DCB=(** . rückbezug, unterparameter, . . . , unterparameter*)

Wird der Name kodiert, so werden die DCB-Angaben von der genannten Datei kopiert. Die Datei muß eine katalogisierte Plattendatei sein und der sie enthaltende Plattenstapel muß geladen sein, bevor die Ausführung des Steps beginnt. (Die Datenträgerfolgenummer und das Schutzdatum werden mit kopiert, es sei denn, sie sind extra auf der DD-Karte kodiert.) Wird ein Rückbezug kodiert, so werden die DCB-Unterparameter von der angezogenen DD-Karte kopiert. (Der Rückbezug kopiert keine DCB-Angaben aus dem Kennsatz der in der angezogenen DD-Karte beschriebenen Datei.) Zusätzlich zum Rückbezug kann jeder beliebige

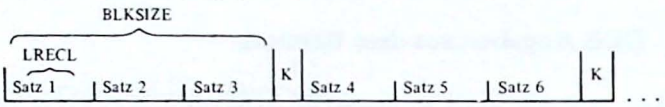
RECFM=F Sätze fester Länge



LRECL die Satzlänge ist für alle Sätze gleich!

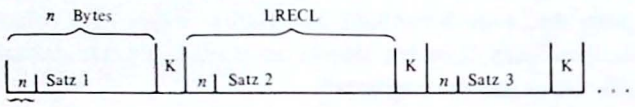
K - Kluft

RECFM=FB Sätze fester Länge geblockt



BLKSIZE die Blocklänge ist ein Vielfaches von LRECL.

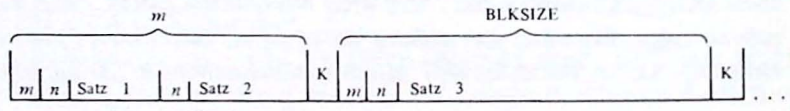
RECFM=V Sätze variabler Länge



4 Bytes $4 \leq n \leq 32{,}756$

LRECL ist die maximale Satzlänge

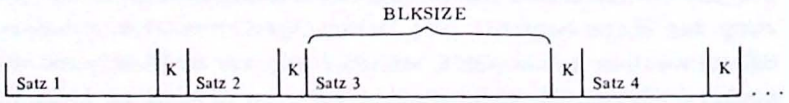
RECFM=VB Sätze variabler Länge geblockt



$8 \leq m \leq 32{,}760$

BLKSIZE muß größer als die Satzlänge oder gleich der Satzlänge der Datei sein.

RECFM=U Sätze undefinierter Länge



BLKSIZE muß größer als die Satzlänge oder gleich der Satzlänge der Datei sein.

Abb. 8.1 Satzformate

DCB-Unterparameter kodiert werden, er überschreibt oder ergänzt die kopierten Unterparameter.

```
//EINGABE  DD  DCB=SYS1.LINKLIB,...
```

Die DCB-Unterparameter werden von der SYS1.LINKLIB kopiert.

```
//EINGABE  DD  DCB=*.DDEINS,...
```

Die DCB-Unterparameter werden von der DDEINS DD-Karte kopiert. Die DD-Karte DDEINS muß im gleichen Step vor der Karte EINGABE liegen.

```
//EINGABE  DD  DCB=(*.STEP1.DDZWEI,BLKSIZE=800,BUFNO=3),...
```

Die DCB-Unterparameter werden von der DDZWEI DD-Karte des Steps STEP1 kopiert. STEP1 muß ein weiter vorne liegender Step sein. Die Unterparameter BLKSIZE und BUFNO im kopierten DCB werden mit den angegebenen überschrieben.

8.6 Name der Datei: DSN

Der Parameter DSN[1]) benennt die Datei. Man unterscheidet zwischen temporären und nicht-temporären Dateien. Eine temporäre Datei wird innerhalb des gleichen Jobs erzeugt und wieder gelöscht. Die nicht-temporäre Datei dagegen kann auch nach Abschluß des Jobs bestehen bleiben. Alle nicht-temporären Dateien müssen einen Namen haben. Für temporäre Dateien ist der Parameter DSN wahlfrei. Ist die Datei neu (DISP=(NEW,..)), so ordnet der Parameter DSN der Datei ihren Namen zu; ist sie dagegen alt (DISP=(OLD,..)), so nennt der Parameter DSN ihren Namen.

8.6.1 Temporäre Dateien

Temporäre Dateien benutzt man als Hilfsspeicher. Falls der Parameter DISP die Datei nicht wieder löscht, wird sie vom System gelöscht. Durch das Löschen einer Banddatei wird die Bandrolle entladen. Das Löschen einer Plattendatei gibt den Platz auf der Platte wieder frei.

[1]) Der Parameter DSN darf auch als DSNAME kodiert werden. Beispielsweise sind DSN=PRIVLIB und DSNAME=PRIVLIB äquivalent.

Eine Datei ist temporär durch das Weglassen des Parameters DSN oder durch das Kodieren von DSN=&&*name*. Wird der DSN-Parameter weggelassen, so ordnet das System der Datei einen nicht zu verwechselnden Namen zu. Jeder folgende Step, der diese Datei benutzt, muß sich auf sie durch einen Rückbezug auf die DD-Karte beziehen. (Parameter DSN=*.rückbezug kann sowohl für eine temporäre als auch nicht-temporäre Datei benutzt werden.)

```
//STEP1     EXEC PGM=EINS
//DDNAME1   DD  UNIT=SYSDA,DISP=(NEW,PASS),...
//STEP2     EXEC PGM=ZWEI
//DDNAME2   DD  DSN=*.STEP1.DDNAME1,DISP=(OLD,PASS)
```

Einer temporären Datei wird ein Name durch das Kodieren von DSN=&&name zugewiesen; die Zeichen && kennzeichnen sie als temporär. Solange der Name nicht mit einem symbolischen Parameter verwechselt werden kann, darf ein einziges Zeichen & kodiert werden (DSN=&name). Der Name ist 1 bis 8 Zeichen lang (siehe Kapitel 9) und muß mit einem alphanumerischen (A-Z, 0-9) oder nationalen Zeichen (@ $ #) beginnen. Die restlichen Zeichen können alphanumerisch oder national sein; Bindestrich (-) und +0 (eine Doppellochung 12 und 0) sind auch gestattet. Auf die temporäre Datei kann man sich später rückbeziehen durch: DSN=&&name oder DSN=*.rückbezug.

```
//STEP1     EXEC PGM=EINS
//DDNAME1   DD  DSN=&&TEMP,...
//STEP2     EXEC PGM=ZWEI
//DDNAME2   DD  DSN=&&TEMP,...
                ODER
//DDNAME2   DD  DSN=*.STEP1.DDNAME1,...
```

8.6.2 Nicht-temporäre Dateien

Nicht-temporäre Dateien sind solche, die nach der Beendigung des Jobs weiterbestehen. Ein zusammen mit der Datei erzeugter Kennsatz nimmt ihren Namen und die DCB-Unterparameter auf. Es ist möglich, die Datei zu katalogisieren, so daß das System außerdem auch die Einheit und die Datenträgernummer des Datenträgers aufzeichnet, auf dem die Da-

tei liegt. Liegt die Datei auf einer Bandrolle, so wird auch noch ihre Stellung auf der Rolle (File Nummer) festgehalten. Obwohl auch eine nicht-temporäre Datei in ein und demselben Step erzeugt und gelöscht werden kann, ist es doch müßig das zu tun. Statt dessen sollte eine temporäre Datei erzeugt werden.

Eine nicht-temporäre Datei wird durch die Kodierung DSN=*name* beschrieben. Der Name besteht aus bis zu 8 Zeichen. Der Name NULLFILE (siehe Kapitel 10) sollte nicht verwendet werden, da er ein Schlüsselwort ist, das der Datei den Status einer Pseudodatei verleiht. Das erste Zeichen des Namens muß alphanumerisch (A-Z, 0-9) oder national (@$#) sein. Die restlichen Zeichen dürfen sein: alphanumerisch, national, Bindestrich oder +0 (12 und 0 Lochung). Oft wird für den Namen der Rückbezug auf eine DD-Karte benutzt, weil dadurch bei einer Namensänderung nur eine DD-Karte betroffen ist.

```
//STEP1     EXEC PGM=ERZEUGEN
//A         DD   DSN=DATEI,...
//STEP2     EXEC PGM=LESEN
//B         DD   DSN=DATEI,...
                 ODER
//B         DD   DSN=*.STEP1.A,...
```

Unter folgenden Umständen darf der Name Sonderzeichen (Leerstelle ,./'()*&+=) enthalten.

1) Der Name wird in Hochkommas eingeschlossen. Ein Hochkomma im Namen muß durch zwei Hochkommas dargestellt werden, z.B. 'O''CLOCK'. Der in Hochkommas eingeschlossene Name darf bis zu 44 Zeichen enthalten.
2) Der Name darf sich nicht auf eine untergliederte Datei, keine index-sequentielle Datei und keine Generation Data Group beziehen.
3) Die Datei ist nicht katalogisiert.
4) Der Name wird in keinem Rückbezug benutzt.
5) Die Datei ist nicht temporär, d.h. DSN=&&name wurde nicht kodiert.
6) Der Dateiname ist nicht qualifiziert.

Dateinamen können *qualifiziert*, d.h. durch das Zusammenhängen von Namen verlängert werden. Jede Namensstufe, *Index* genannt, muß der oben beschriebenen Namenskonvention entsprechen. Die Namensstufen werden durch einen Punkt voneinander getrennt, z.B. A.B.Z. Qualifi-

zierte Namen dürfen bis zu 44 Zeichen, die Punkte mitgerechnet, enthalten. (Bei dem Namen einer Generation Data Group ist die Länge auf 35 Zeichen beschränkt.) Auf diese Art und Weise sind 22 Qualifikationsstufen möglich. Qualifizierte Namen werden benutzt, um Dateien zu Gruppen zusammenzuschließen bzw. um die Namen unverwechselbar zu gestalten. Beispielsweise ist es sehr wahrscheinlich, daß der Name DATA.A27.B74276 nicht auch noch ein zweites Mal vorkommt. Die Dateien SCHMITT.SOURCE,SCHMITT.DATEN und SCHMITT.BERICHT bilden die dem Programmierer SCHMITT gehörende Dateiengruppe.

```
//EIN      DD  DSN=SCHMITT.A2267.TEST6,...
//AUS      DD  DSN=ABT.GR4.A707.DATEI32,...
```

Für eine neue Datei mit qualifizierten Namen muß ein Index für jede Indexstufe eingerichtet werden. Das IBM-Utility-Programm IEHPROGM, das in Kapitel 14 beschrieben wird, baut diese Indizes auf. Beispielsweise müssen für die Datei A.B.C die Indexstufen A und A.B aufgebaut worden sein, bevor diese Datei erzeugt und katalogisiert werden kann.

8.6.3 Andere Dateien

Der Parameter DSN erfährt für untergliederte Dateien, für Generation Data Group und index-sequentielle Dateien eine kleine Abwandlung. (Diese Dateitypen werden in späteren Kapiteln genauer besprochen.) Die allgemeine Form des Parameters ist DSN=name(wert) bzw. DSN=&&name(wert), je nachdem ob es sich um eine nicht-temporäre oder temporäre Datei handelt. Der Name ist der Dateiname, und der Parameter wert ist vom Dateityp abhängig.

Um sich auf das Glied einer untergliederten Datei zu beziehen, wird DSN=name(glied) kodiert. Die Angabe in der Klammer ist der Name des Gliedes. Beispielsweise ist mit DSN=SYS1.PROCLIB(DRUCKER) das Glied DRUCKER der Datei SYS1.PROCLIB gemeint. Um sich auf eine bestimmte Generation einer Generation Data Group zu beziehen, kodiert man DSN=name(zahl). Zahl ist die relative Generationsnummer; ihr Wert ist 0 oder eine mit Vorzeichen versehene ganze Zahl. Beispielsweise bezieht sich DSN=STEUER(-1) auf die relative Generation -1 der Dateigruppe STEUER. Eine index-sequentielle Datei kann aus dem Index-, dem Hauptdaten- (PRIME) und dem Überlauf-Bereich (OVERFLOW) bestehen. Diese Bereiche werden mit DSN=name(INDEX), DSN=name (PRIME) und DSN=name (OVFLOW) angesprochen.

8.7 Ein-/Ausgabe-Einheit: UNIT

Eine Ein-/Ausgabe-Einheit ist ein bestimmtes E/A-Gerät, beispielsweise: Eine 2311-Platteneinheit, eine 2400-Bandeinheit, ein 1403-Drukker etc. Das vom UNIT-Parameter verlangte E/A-Gerät darf nicht mit dem Datenträger verwechselt werden. Unter einem Datenträger (VOLUME) versteht man: Eine Magnetbandrolle, eine Trommel, ein Plattenstapel, eine Unterzelle in einer Datenzelle (Streifenspeicher) oder den Teil einer Plattenspeichereinheit IBM 2302, der von einem Zugriffsmechanismus versorgt wird. Das bedeutet, daß ein Datenträger auf ein E/A-Gerät geladen werden muß. Ein bestimmter Datenträger wird mit dem Parameter VOL angefordert. Er wird später in diesem Kapitel beschrieben.

Der Parameter UNIT verlangt das E/A-Gerät durch seine Hardware-Adresse, den Geräte-Typ, die Geräte-Gruppe oder dadurch, daß das gleiche E/A-Gerät verlangt wird, das von einer anderen Datei im gleichen Step benutzt wird. Der Parameter UNIT hat Unterparameter. Diese ermöglichen:

- Das parallele Laden mehrerer Datenträger einer Datei,
- das Verzögern der Lademeldung bis zu dem Zeitpunkt, zu dem die Datei eröffnet wird (Makro OPEN im Programm) und
- die Wahl eines E/A-Gerätes, das mit dem von einer anderen Datei benutzten E/A-Gerät identisch bzw. nicht identisch ist.

Der Parameter UNIT ist stets nötig, es sei denn, die Datei ist katalogisiert oder sie wurde herübergereicht (DISP=(...,PASS,..). In der folgenden Liste werden die Fälle aufgezählt, in denen UNIT nicht kodiert werden muß:

1) SYSOUT wird in der DD-Karte kodiert. UNIT ist wahlfrei. Läßt man den Parameter weg, so wird ein in der katalogisierten Reader-Prozedur festgehaltener Wert unterstellt.
2) DUMMY wird in der DD-Karte kodiert. Wieder ist UNIT wahlfrei. Wird es jedoch kodiert, so wird dieser Parameter ignoriert.
3) Eine Datei besteht bereits und ist katalogisiert. Gleichwertig damit ist die bestehende Datei, die von einem Step weitergereicht wurde. (Auch der Parameter VOL=SER braucht nicht mehr kodiert zu werden.)

4) Einer Datei werden mit dem Parameter VOL=REF= Datenträger zugeordnet, die von einer früheren Datei im gleichen Step benutzt werden.
5) Eine Datei teilt ihren Platz mit einer früheren Datei durch die Platzanforderungsparameter SUBALLOC oder SPLIT.

Um ein E/A-Gerät anzufordern, kodiert man UNIT=*adresse*, UNIT=*typ* oder UNIT=*gruppe*. Mit Adresse ist die Hardware-Adresse gemeint. Unter Typ versteht man die numerische Bezeichnung des Geräte-Typs. Gruppe ist der in einer Installation einer Gruppe von E/A-Geräten zugeordnete Name. Um das von einer anderen Datei benutzte E/A-Gerät anzufordern, wird UNIT=AFF=*ddname* kodiert. Die Angabe ddname bezieht sich auf eine frühere DD-Karte im gleichen Step.

8.7.1 Hardware-Adresse

Mit UNIT=*adresse* wird ein bestimmtes E/A-Gerät mit Hilfe einer dreistelligen Hardware-Adresse angefordert. Diese Hardware-Adressen werden bei der Installation der Anlage festgelegt. Verlangt man beispielsweise die Bandeinheit 0C4, so muß UNIT=0C4 kodiert werden. Die Hardware-Adressen sollten allerdings nicht benutzt werden, es sei denn unter besonderen Umständen. Im Multiprogramming-Betrieb könnte die verlangte Einheit nämlich bereits einem anderen Job zugeordnet worden sein. Es ist besser, man fordert ein Gerät aus einer Gruppe an, und überläßt es dem System, ein in der Gruppe verfügbares auszuwählen.

Um die Unterzelle einer 2321-Datenzelle anzufordern, kodiert man UNIT=adresse/unterzelle. Die Unterzelle ist eine der Ziffern 0-9. Z.B. fordert UNIT=30D/2 die Unterzelle 2 der 2321 mit der Hardware-Adresse 30D an. Im Gegensatz dazu verlangt UNIT=30D irgendeine verfügbare Unterzelle der 2321.

8.7.2 E/A-Geräte-Typ

Um ein bestimmtes E/A-Gerät mit Hilfe der IBM-Modellbezeichnung anzufordern, kodiert man UNIT=*typ*. Jedes IBM-Gerät hat automatisch eine vorher festgelegte Modellbezeichnung. Z.B. wird mit UNIT=2311 eine 2311 Plattenspeichereinheit angefordert. Die folgende Tabelle führt die verfügbaren Gerätetypen auf.

Tabelle 1 Gerätetypen

Bandeinheiten

Typ	Gerät
2400	9-Spur Magnetbandeinheit der Serie 2400 mit 800 bpi Schreibdichte.
2400-1	Magnetbandeinheit der Serie 2400 mit 7-Spur Verträglichkeit und ohne Datenumsetzung.
2400-2	Magnetbandeinheit der Serie 2400 mit 7-Spur Verträglichkeit und Datenumsetzung.
2400-3	9-Spur Magnetbandeinheit der Serie 2400 mit 1600 bpi Schreibdichte.
2400-4	9-Spur Magnetbandeinheit der Serie 2400 mit 800 und 1600 bpi Schreibdichte.
2420-5	Eine Bandeinheit 2420 Modell 5 mit 1600 bpi Schreibdichte.

Speichereinheiten mit direktem Zugriff

2301	Trommelspeichereinheit 2301
2302	Plattenspeichereinheit 2302, ein Laufwerk
2303	Plattenspeichereinheit 2303
2311	Plattenspeichereinheit 2311
2314	Plattenspeichereinheit 2314 mit mehreren Laufwerken
2321	Streifenspeicher 2321

Konventionelle Geräte

1052	Drucker mit Tastatur 1052
1403	1403 bzw. 1404 Drucker
1442	1442 Karteneinheit
1443	1443 Drucker
2501	2501 Kartenleser
2520	2520 Karteneinheit
2540	2540 Karteneinheit, Lesestation
2540-2	2540 Karteneinheit, Stanzstation
2671	2671 Lochstreifenleser
2741	2741 Datenstation

Typ	Gerät
Geräte für die graphische Verarbeitung	
1053	1053 Drucker Modell 4
2250-1	2250 Sichtgerät, Modell 1
2250-3	2250 Sichtgerät, Modell 3
2260-1	Optische Anzeige 2260, Modell 1 (lokaler Anschluß)
2260-2	Optische Anzeige 2260, Modell 2 (lokaler Anschluß)
2280	2280 Filmaufzeichengerät
2282	2282 Filmeinheit
2760	2760 Bildgerät
1130	Rechner 1130

8.7.3 E/A-Geräte-Gruppe

Um ein E/A-Gerät aus einer Geräte-Gruppe der Installation anzufordern, kodiert man UNIT=*gruppe*. UNIT=SYSDA ist traditionsgemäß die Gruppe der Speicher mit direktem Zugriff. UNIT=SYSSQ sind Einheiten, die sequentielle Dateien aufnehmen können. In einer Installation könnte man eine Geräte-Gruppe zur Aufnahme temporärer und eine andere zur Aufnahme nicht-temporärer Plattendateien einrichten. Beide Gruppen könnten verschiedene E/A-Geräte enthalten. Das folgende Beispiel zeigt den Unterschied zwischen Hardware-Adresse, Geräte-Typ und Geräte-Gruppe.

	Verfügbare Geräte			
	2311#1	2311#2	2314#1	2314#2
Adresse	130	131	230	231
Typ	2311	2311	2314	2314
Gruppe	SYSDA	SYSDA	SYSDA	SYSDA
	SCRATCH	SCRATCH	SCRATCH	

Mit UNIT=130 wird nur die Einheit 2311#1 ausgewählt. UNIT=2311 dagegen fordert entweder die 2311#1 oder die 2311#2 an. UNIT=SYSDA stellt irgendeines der 4 Geräte zur Verfügung. Wird UNIT=SCRATCH kodiert, so wird die Einheit 2311#1, 2311#2 oder die Einheit 2314#1 ausgewählt. Falls ein Datenträger in einem System mit der Einrichtung AVR (Automatic Volume Recognition) geladen wird, fordert UNIT=gruppe den Operator auf, den Datenträger auf einem bestimmten Gerät zu montieren. UNIT=typ verlangt vom Operator, den Datenträger auf irgendeinem der verfügbaren Geräte des betreffenden Typs zu montieren.

8.7.4 Einheiten-Affinität

Kodiert man UNIT=AFF=*ddname*, so werden Dateien auf ladbaren Datenträgern dem gleichen E/A-Gerät zugeordnet, das bereits von einer anderen Datei benutzt wird. Der Wert ddname bezieht sich auf den Namen einer vorausgegangenen DD-Karte des gleichen Steps. Die Einheiten-Affinität spart E/A-Geräte ein, indem sie zwei Dateien auf das gleiche physische Gerät lädt. Das E/A-Gerät wird nacheinander von beiden Dateien benutzt. Das bedeutet, jede Datei muß abgeschlossen worden sein, bevor die nächste Datei eröffnet wird. Die Einheiten-Affinität schließt das verzögerte Laden des Datenträgers mit ein. (Für den Datenträger wird erst beim Eröffnen der Datei eine Lademeldung ausgegeben.) Die Einheiten-Affinität kann natürlich nur für E/A-Geräte mit auswechselbaren Datenträgern verlangt werden. Das sind 2311 und 2314 Plateneinheiten, 2400 Bandgeräte und 2321 Streifenspeichereinheiten.

Wird für den 2321 Streifenspeicher Affinität verlangt, so werden die Dateien der gleichen Einheit, aber möglicherweise verschiedenen Zellen zugeordnet. Bezieht sich der Wert ddname auf eine DD-Karte mit DUMMY oder DSN=NULLFILE, so wird der die Affinität fordernden DD-Karte ebenfalls der Status einer Pseudodatei verliehen.

```
//STEP1    EXEC PGM=PROGM1
//A        DD   UNIT=2314,...
//B        DD   UNIT=AFF=A,...
//C        DD   UNIT=AFF=D,...
```

Falsch! Der Wert ddname muß sich auf eine frühere DD-Karte beziehen.

```
//D        DD   UNIT=DISK,...
```

8.7.5 Besondere Optionen

Mehrere Datenträger, verzögertes Laden und Einheitentrennung von anderen Dateien kann mit dem UNIT-Parameter angefordert werden.

```
        typ      P
        gruppe   volumes
UNIT=(------,-------,DEFER,SEP=(ddname , . . . , ddname))
```

8.7.5.1 *Paralleles Laden von Datenträgern*

Falls eine Datei sich über mehr als einen Datenträger erstreckt, kann das Laden aller Datenträger zur gleichen Zeit durch das Kodieren von UNIT=(gerät,volumes,...) verlangt werden. Der Wert gerät kann ein Geräte-Typ oder eine Geräte-Gruppe sein. Der Wert volumes ist die Anzahl der parallel zu ladenden Datenträger (2-59). Die Angabe für volumes darf die Anzahl der verfügbaren Laufwerke nicht überschreiten.

Das parallele Laden spart Zeit ein, da für den Operator die Notwendigkeit entfällt, die Datenträger nacheinander auf einer einzigen Einheit zu laden und zu entladen. Allerdings sperrt das parallele Laden im Multiprogramming-Betrieb die Einheiten für andere Jobs. Das parallele Laden gestattet auch Dateien zu verarbeiten, die sich über mehrere Plattenstapel erstrecken. Nur ein E/A-Gerät wird zugeordnet, falls die Angabe volumes gleich 0 ist, oder falls sie weggelassen worden ist.

Ist die Datei katalogisiert, oder gibt der Parameter VOL die Nummern aller Datenträger an, so wird P statt volumes kodiert. Der Wert für volumes wird in diesem Fall dem Katalog oder dem VOL-Parameter entnommen. Der Geräte-Typ oder die Geräte-Gruppe sollte für katalogisierte Dateien weggelassen werden, weil diese Information im Katalog enthalten ist.

```
//A        DD  UNIT=(2311,2),...
```

Zwei 2311 Einheiten werden parallel geladen.

```
//B        DD  UNIT=(,P),...
```

Alle Datenträger der katalogisierten Datei werden parallel geladen.

```
//C        DD  UNIT=(TAPE,P),VOL=SER=(200,300),...
```

Die beiden Bandrollen 200 und 300 werden parallel geladen.

8.7.5.2 *Verzögertes Laden eines Datenträgers: DEFER*

Um das Laden eines Datenträgers bis zum Eröffnen der Datei zurückzustellen, kodiert man UNIT=(gerät,,DEFER,...). Das System wird die Lademeldung für den Datenträger erst beim Eröffnen der Datei ausgeben. DEFER wird benutzt, um das unnütze Laden eines Datenträgers zu verhindern, wenn er in einem bestimmten Lauf nicht erforderlich ist. DEFER wird ignoriert, falls der Datenträger bereits geladen ist. Darüber hinaus darf DEFER für neue Plattendateien nicht gegeben werden,

weil die Platzzuordnung bei der Stepinitialisierung erfolgen muß. DEFER kann ebensowenig für index-sequentielle Dateien gegeben werden.

```
//A          DD  UNIT=(TAPE,,DEFER),...
//B          DD  UNIT=(DISK,2,DEFER),DISP=OLD,...
//C          DD  UNIT=(,P,DEFER),...
```

8.7.5.3 *Trennung von Dateien: SEP*

Um eine Plattendatei von einem Zugriffsmechanismus fernzuhalten, der bereits von anderen Dateien benutzt wird, kodiert man UNIT=(gerät,...,SEP=(ddname,...,ddname)). Da SEP ein Schlüsselwortparameter ist, wird er nach dem letzten Positions-Unterparameter kodiert. Mit den Werten ddname können bis zu 8 DD-Karten im gleichen Step angesprochen werden, von denen die Datei ferngehalten werden soll.

Der Parameter SEP schränkt den Kampf um den Zugriffsmechanismus ein, indem er gleichzeitig benutzte Dateien auf verschiedene E/A-Geräte mit getrennten Zugriffsmechanismen legt. Auf diese Art und Weise kann die Verarbeitungszeit beträchtlich verkürzt werden. Enthält beispielsweise der gleiche Plattenstapel eine Eingabe- und eine Ausgabedatei, so muß der Zugriffsarm bei jedem Lesen und Schreiben bewegt werden. Sind die Dateien dagegen getrennt, so entfällt dieser Streit um den Zugriffsarm. (Im Multiprogramming-Betrieb kann es trotzdem mit den Dateien anderer Jobs Streit um den Zugriffsarm geben, aber das liegt außerhalb der Kontrolle des Benutzers, es sei denn, er fordert einen privaten Datenträger an.)

```
//STEP1      EXEC PGM=EINS
//A          DD  UNIT=SYSDA,...
//B          DD  UNIT=(SYSDA,SEP=A),...
//C          DD  UNIT=(SYSDA,SEP=D),...
```

Falsch! Der Rückbezug darf nur auf eine frühere DD-Karte erfolgen.

```
//D          DD  UNIT=(SYSDA,3,DEFER,SEP=(A,B))
```

Die Fortsetzung des SEP-Unterparameters auf einer Folgekarte ist möglich, wenn er nach einem vollständigen ddname, das Komma eingeschlossen, unterbrochen wird.

Kann die SEP-Anforderung nicht befriedigt werden, so wird sie von einem PCP-System ignoriert. MFT- und MVT-Systeme dagegen fragen

beim Operator an, ob der Job abgebrochen (Cancel) oder ob die SEP-Anforderung ignoriert werden soll. Sollte sich der Wert ddname auf eine Pseudodatei beziehen (DUMMY oder DSN=NULLFILE), so wird die Dateientrennung für diesen ddname ebenfalls ignoriert. Besitzt das System die Einrichtung AVR (Automatic Volume Recognition, automatische Datenträger Registrierung) und ist der Datenträger bereits geladen, so wird der Unterparameter SEP auch in diesem Fall ignoriert. (Das System verlangt vom Operator kein Entladen eines Datenträgers, nur um eine SEP-Anforderung zu erfüllen.) SEP kann dazu dienen, um das Laden von 2321 Datenzellen auf getrennten Geräten zu erreichen. Normalerweise lädt das System sie auf so wenig Geräten wie möglich.

8.8 Disposition der Datei: DISP

Der Parameter DISP beschreibt den gerade vorhandenen Status der Datei (bestehend, neu, soll modifiziert werden). Gleichzeitig legt er für das System die Disposition der Datei (weiterreichen, festhalten, katalogisieren oder löschen) bei normalem oder abnormalem Stepende fest. Der Parameter DISP ist stets nötig, es sei denn, die Datei wird im gleichen Step erzeugt und wieder gelöscht. Die allgemeine Form des Parameters DISP ist:

DISP=(*status, normale disposition, abnormale disposition*)
Die verschiedenen Möglichkeiten lauten:

```
            KEEP
      NEW   DELETE    KEEP
      MOD   PASS      DELETE
      OLD   CATLG     CATLG
      SHR   UNCATLG   UNCATLG
DISP=(---- , --------- ,----------)
```

8.8.1 Status der Datei

Der Status der Datei, nämlich NEW, MOD, OLD oder SHR, beschreibt ihren Zustand zu Beginn des Jobsteps. Falls die Datei neu ist (NEW), erzeugt das System einen Kennsatz. Besteht sie dagegen bereits (OLD), so lokalisiert das System die Datei und liest ihren Kennsatz.

8.8.1.1 *Neue Dateien: NEW*

DISP=(NEW,...) erzeugt eine neue Datei. Man benötigt dazu außerdem den Parameter UNIT. Der Parameter VOL ist nur notwendig, wenn sie auf einen bestimmten Datenträger kommen soll. Bei Dateien auf Speichern mit direktem Zugriff ist außerdem der Parameter SPACE nötig. Der Step wird abnormal beendet, sollte sich auf dem gleichen Datenträger bereits eine Datei des gleichen Namens befinden.

Wird der erste Parameter nicht kodiert, so wird NEW unterstellt. Z.B. ist DISP=(,KEEP) mit DISP=(NEW,KEEP) gleichbedeutend. NEW wird ebenfalls unterstellt, falls DISP völlig weggelassen wird.

8.8.1.2 *Modifizierte Dateien: MOD*

DISP=(MOD,...) modifiziert eine sequentielle Datei. Beim Eröffnen der Datei positioniert MOD den Lese-/Schreibmechanismus hinter den letzten Satz der Datei. Dadurch werden Hinzufügungen zu dieser sequentiellen Datei ermöglicht.

Besteht die Datei noch nicht, so ändert das System eine Disposition von MOD in NEW ab, vorausgesetzt der Parameter VOL verlangt keinen bestimmten Datenträger. Wurde VOL dagegen kodiert, so erwartet das System, die Datei auf dem angegebenen Datenträger zu finden und bricht den Step ab, falls es die Datei dort nicht finden kann. Wurde der Parameter VOL dagegen nicht kodiert, so prüft das System, ob die Datei katalogisiert oder hinübergereicht (PASS) ist. Ist keines von beiden der Fall, so wird angenommen, daß die Datei nicht existiert, und das System erzeugt sie so als wenn NEW kodiert worden wäre.

Eine neue Datei bekommt ihren Datei-Ende-Anzeiger erst, wenn sie als Ausgabedatei eröffnet und geschlossen worden ist. Vorher besitzt sie keinen solchen. Manchmal ordnet man für eine Plattendatei Platz zu, in der Erwartung, daß spätere Jobs mit der Disposition MOD Daten hinzufügen werden. Der Versuch, eine solche Datei zu lesen, wird nicht bestimmbare Ergebnisse liefern, es sei denn, die Datei wurde bereits einmal als Ausgabedatei eröffnet und auch wieder geschlossen, so daß der Datei-Ende-Anzeiger gesetzt ist.

MOD wird benutzt, um Daten zu einer Datei hinzuzufügen, die sich bereits über mehrere Datenträger erstreckt. Die Erzeugung dieser sich über mehrere Datenträger erstreckenden Datei geschah ebenfalls mit

MOD. Im Zusammenhang mit MOD sollte man stets CATLG spezifizieren, selbst wenn die Datei bereits katalogisiert ist. Die Angabe CATLG bewirkt, daß zusätzliche Datenträgernummern in den Katalog aufgenommen werden. Falls die Datenträger der Datei noch nicht geladen sind, sollte man die Datenträgeranzahl im VOL-Parameter kodieren bzw. mit dem UNIT-Parameter verzögertes Laden anfordern, damit das System das Laden und Entladen der Datenträger durchführt.

Für eine mit MOD zu verlängernde Datei sollten in der DD-Karte, die diese Verlängerung durchführt, keine DCB-Angaben kodiert werden, da diese ja im Kennsatz enthalten sind. Es besteht nämlich die Gefahr, daß die DCB-Angaben falsch, d.h. nicht übereinstimmend mit denen im Kennsatz, kodiert werden. Falls man DCB-Angaben in der DD-Karte kodiert, muß man sicherstellen, daß sie mit denen der Datei übereinstimmen.

8.8.1.3 *Bestehende Dateien: OLD*

DISP=(OLD,..) beschreibt die bestehende Datei. Sie kann eine Eingabedatei sein bzw. eine noch unvollständige Ausgabedatei. Der Step bekommt mit dieser Disposition das alleinige Verfügungsrecht über diese Datei. (Im Multiprogramming-Betrieb haben verschiedene Jobs die Möglichkeit, konkurrierend die gleiche Plattendatei zu lesen.)

Ist die bestehende Datei katalogisiert bzw. wurde sie von einem früheren Step herübergereicht, so ist der Parameter DSN gewöhnlich der einzige andere DD-Parameter, der nötig ist. (Für Magnetbandrollen kann der Parameter LABEL noch nötig sein.) Ist die bestehende Datei dagegen nicht katalogisiert und wurde sie auch nicht von einem früheren Step herübergereicht, so müssen die Parameter UNIT und VOL kodiert werden.

8.8.1.4 *Gemeinsame Benutzung von Dateien: SHR*

Die Disposition DISP=(SHR,..) gestattet es, bestehende Dateien gemeinsam zu benutzen. SHR ist mit OLD identisch, mit der einen Ausnahme, daß im Multiprogramming-Betrieb mehrere Jobs die Datei konkurrierend benutzen können. SHR darf nur für Eingabedateien verwendet werden. SHR ist in MFT- und MVT-Systemen sehr nötig, denn öffentliche Bibliotheken, wie die SYS1.LINKLIB oder wie Unterprogrammbibliotheken sollten für jeden Job im System zur Verfügung stehen. Wird SHR in einem PCP-System kodiert, so wird es wie OLD behandelt. SHR sollte

für alle Eingabedateien benutzt werden. (Falls eine Datei von einem anderen Job verändert wird, wird dieser die Disposition OLD verwenden, um die gemeinsame Benutzung der Datei zu verhindern.

```
//STEP1     EXEC PGM=PROGM1
//A         DD   DSN=SAETZE,DISP=(,PASS),UNIT=2311,SPACE=(80,100)
```

Die Datei SAETZE wird erzeugt. Da sie von jetzt ab besteht, muß man sich bei späterem Gebrauch auf sie mit MOD, SHR oder OLD beziehen.

```
//STEP2     EXEC PGM=PROGM2
//B         DD   DSN=DATEI2,DISP=(MOD,CATLG),UNIT=2314,
//               SPACE=(80,100)
```

Falls DATEI2 nicht im Katalog gefunden wird, nimmt das System an, daß diese Datei nicht besteht und erzeugt sie deshalb. Falls sie jedoch bereits besteht, werden neue Daten nach dem letzten Satz hinzugefügt.

```
//C         DD   DSN=LIB,DISP=(MOD,CATLG),UNIT=2314,VOL=SER=DA0010
```

Die Datei LIB muß eine bestehende Datei sein. Neue Daten werden hinter dem letzten Satz angefügt.

```
//STEP3     EXEC PGM=PROGM3
//D         DD   DSN=SYS1.FORTLIB,DISP=OLD
```

SYS1.FORTLIB ist eine bestehende Datei. STEP3 hat alleinigen Zugriff zu ihr. Falls das System sie im Katalog nicht auffinden kann, wird dieser Step abnormal beendet.

```
//E         DD   DSN=SYS1.FORTLIB,DISP=SHR
```

Der einzige Unterschied zwischen D und E ist, daß E auch anderen Jobs gestattet, SYS1.FORTLIB zu lesen.

8.8.2 Normale Disposition

Die normale Disposition, der zweite Ausdruck im DISP-Parameter, gibt die Disposition der Datei an, wenn der Job normal endet. Die normale Disposition kann weggelassen werden, wenn sich der Status der

Datei nicht ändern soll, bestehende Dateien (MOD,OLD,SHR) also bestehen bleiben; Dateien mit NEW werden wieder gelöscht.

Die Dispositionen für Dateien auf einem Speicher mit direktem Zugriff sind verschieden von denen auf Magnetbändern. Plattenspeicherplatz muß unversehrt bleiben, wenn die Datei festgehalten wird (KEEP). Der Platz kann für anderweitige Verwendung wieder freigegeben werden, wenn die Datei gelöscht wird. Das Festhalten und das Löschen von Magnetbanddateien unterscheidet sich nicht voneinander; das Band wird zurückgespult und entladen. Anschließend wird dem Operator mitgeteilt, ob das Band festgehalten werden muß oder freigegeben werden kann. Der Operator wird ein festzuhaltendes Band aufbewahren, während er ein zu löschendes Band freigeben wird. Für die Daten auf dem Band macht das keinen Unterschied, ihnen geschieht nichts. (Das Makro CLOSE der Assembler-Sprache kann die normale Disposition für Bänder überschreiben.)

Wird eine Plattenspeicherdatei zwischen Jobsteps hinübergereicht, so bleibt sie auf der Platte erhalten. Werden Banddateien hinübergereicht, so wird das Band zwar zurückgespult, aber nicht entladen.

Unter folgenden Umständen wird die Disposition nicht ausgeführt:

1) Wegen JCL-Fehler wird der Step nicht initialisiert.
2) Wegen eines Parameters COND in der JOB- oder EXEC-Karte wird der Step übergangen. Nur für hinübergereichte Dateien wird die Disposition ausgeführt.
3) Eine Banddatei ohne Angabe von VOL=SER oder eine Plattendatei mit verzögertem Laden (UNIT=(..,,DEFER)) wurde nicht eröffnet.
4) Der Step endet abnormal nach der Gerätezuordnung und vor der Stepausführung, weil Plattenspeicherplatz nicht verfügbar ist. Bestehende Dateien (OLD,MOD,SHR) bleiben bestehen, Dateien mit NEW werden wieder gelöscht.

8.8.2.1 *Weiterreichen von Dateien: PASS*

Durch DISP=(...,PASS,...) wird die Datei den nachfolgenden Steps übergeben. Jeder der nachfolgenden Steps kann die Datei einmal benutzen. Die Disposition PASS spart Rüstzeiten ein, denn das System bewahrt sich die Informationen der Datei über Lage und Datenträger auf. Außerdem bleibt ein ladbarer Datenträger, der eine solche Datei enthält, geladen. Beide Dateiarten, die temporäre und die nicht-temporäre Datei können auf diese Weise übergeben werden.

Die endgültige Dispositionsangabe wird den nachfolgenden Steps überlassen. Falls auf die Datei in einem nachfolgenden Step kein Bezug erfolgt, wird die Datei weiterhin übergeben. Wird PASS im letzten Step kodiert, oder erfolgt keine endgültige Dispositionsvergabe, so werden temporäre Dateien gelöscht, während nicht-temporäre Dateien ihren anfänglichen Status wieder annehmen - d.h. bestehende Dateien (MOD, OLD, SHR) fahren fort zu bestehen; Dateien mit NEW werden gelöscht.

Durch PASS wird die Dateifolgenummer einer Banddatei nicht übergeben. Ist die Datei nicht die erste Datei auf einem Band, so muß der Parameter LABEL angegeben werden, um die Dateifolgenummer zu spezifizieren. Sollten mehrere Dateien in einem Job den gleichen Namen haben, so kann nur eine von ihnen übergeben werden. Nachfolgende Steps können durch den Namen oder durch einen Rückbezug auf eine übergebene Datei Bezug nehmen. Die Parameter UNIT und VOL müssen nicht kodiert werden, denn diese Information wird bei der Disposition PASS mit übergeben.

```
//STEP1    EXEC PGM=PROGM1
//A        DD   DSN=&&TEMP,DISP=(,PASS),UNIT=2311,VOL=SER=DA0400,
//             SPACE=(80,100)
//STEP2    EXEC PGM=PROGM2
//B        DD   DSN=&&TEMP,DISP=(OLD,PASS)
                ODER
//B        DD   DSN=*.STEP1.A,DISP=(OLD,PASS)
```

8.8.2.2 *Halten von Dateien: KEEP*

Die Angabe DISP=(...,KEEP,...) hält nicht-temporäre Dateien. Eine Datei auf einem Plattenspeicher bleibt unverändert erhalten; bei einer Datei auf einem Magnetband wird das Band zurückgespult und entladen. Liegt die Datei auf einem ladbaren Datenträger, so wird für den Operator eine KEEP-Meldung ausgegeben. Sollte für eine temporäre Datei die Disposition KEEP kodiert worden sein, so wird diese Disposition in PASS abgeändert. (Die Datei wird gelöscht, falls der Parameter DSN nicht kodiert wurde und falls DEFER im UNIT-Parameter angegeben wurde.) Wird KEEP zusammen mit NEW verwendet, so wird der Dateikennsatz zusammen mit der Datei gehalten.

```
//AUSDD    DD   DSN=STAMMDAT,DISP=(,KEEP),UNIT=2314,
//             VOL=SER=DA0300,SPACE=(80,100)
```

Die Datei STAMMDAT wird erzeugt und nach der Beendigung des Steps festgehalten. Da STAMMDAT weder katalogisiert noch übergeben wurde, müssen bei jeder Benutzung der Datei in einem nachfolgenden Step oder in einem späteren Job die Angaben UNIT und VOL gemacht werden.

```
//SPAETER  DD  DSN=STAMMDAT,DISP=(OLD,KEEP),UNIT=2314,
//             VOL=SER=DA0300
```

Das System lokalisiert STAMMDAT auf dem angegebenen Datenträger und dem angegebenen Gerät. Um auf eine bestehende Datei Bezug zu nehmen, muß OLD, MOD oder SHR angegeben werden. Hätte man NEW statt OLD benutzt, so hätte das System den Step abgebrochen, weil auf dem Datenträger bereits eine Datei gleichen Namens existiert.

8.8.2.3 *Löschen von Dateien: DELETE*

DISP=(...,DELETE,...) löscht Dateien. Der von einer Plattendatei belegte Speicher wird freigegeben. Bei einer Banddatei wird das Band zurückgespult und entladen. Der Operator erhält eine Nachricht, daß die Banddatei gelöscht wurde. (Im Gegensatz zur Plattendatei geschieht mit den Daten des Bandes nichts.) Wurde die Datei über den Katalog lokalisiert (UNIT und VOL waren nicht kodiert worden), so wird die Datei gleichzeitig entkatalogisiert. Wird DISP=(SHR,DELETE) kodiert, so unterstellt das System DISP=(OLD,DELETE).

8.8.2.4 *Katalogisieren von Dateien: CATLG*

DISP=(..,CATLG,..) katalogisiert nicht-temporäre Dateien. Die Angabe CATLG ist der Angabe KEEP ähnlich. Zusätzlich allerdings werden die Angaben UNIT und VOL zusammen mit dem Dateinamen im Katalog festgehalten. Die Dateifolgenummer für eine Banddatei wird ebenfalls im Katalog festgehalten, nicht dagegen der Typ der Bandkennsätze und genau so wenig die Schreibdichte. (Das System unterstellt Standardkennsätze; der Anwender braucht also den Parameter LABEL nur für Nicht-Standard-Kennsätze zu kodieren.)

Die Angaben UNIT und VOL sind für die Funktion CATLG stets erforderlich, es sei denn, die Datei ist bereits katalogisiert. Falls die Datei bereits katalogisiert ist, wirkt sich die Disposition CATLG genau so wie PASS aus. Versucht man CATLG für temporäre Dateien zu ko-

dieren, so wird diese Disposition in PASS abgeändert. (Die Datei wird gelöscht, falls der Parameter DSN nicht kodiert wurde und falls DEFER im UNIT-Parameter angegeben wurde.) Für eine Datei über mehrere Datenträger werden alle Datenträgernummern im Katalog festgehalten. Wird eine Datei, die sich auf mehreren Datenträgern befindet, noch einmal verlängert, so bewirkt die Disposition DISP=(MOD,CATLG), daß auch die zusätzlichen Datenträgernummern im Katalog eingetragen werden.

Ist der Dateiname qualifiziert (z.B. A.B.C), so muß das in Kapitel 14 beschriebene IBM-Utility-Programm IEHPROGM benutzt werden, um im Katalog für jede höhere Indexstufe einen Eintrag zu erzeugen.

```
//KARTEN   DD   DSN=KARTDAT,DISP=(,CATLG),UNIT=2400,VOL=SER=000500
```

Die Datei KARTDAT wird erzeugt und katalogisiert. In einer späteren DD-Karte kann man sich auf sie mit ihrem Namen unter Weglassung der Angaben UNIT und VOL beziehen.

```
//SPAETER  DD   DSN=KARTDAT,DISP=(OLD,CATLG)
                ODER
//SPAETER  DD   DSN=KARTDAT,DISP=(OLD,KEEP)
```

Wenn die Datei bereits katalogisiert ist, sind CATLG und KEEP äquivalent. Jedoch sollte man bei einer Datei, die sich über mehrere Datenträger erstreckt und die wieder erweitert wird, CATLG verwenden, um die zusätzliche Datenträgernummer ebenfalls im Katalog aufzunehmen.

```
//CATLG    DD   DSN=ETAT,DISP=(OLD,CATLG),UNIT=2400,VOL=SER=000300
```

Durch eine solche DD-Karte kann eine bestehende Datei katalogisiert[2]) werden.

8.8.2.5 *Entkatalogisieren von Dateien: UNCATLG*

DISP=(..,UNCATLG,..) entkatalogisiert eine Datei. UNCATLG hat die gleiche Funktion wie KEEP, nur daß der Dateiname außerdem noch aus dem Katalog entfernt wird. Ist die Datei nicht katalogisiert, so ist UNCATLG mit KEEP identisch.

---

[2]) Anmerkung des Übersetzers: Man nimmt dazu am zweckmäßigsten das "Null"-Programm IEFBR14.

```
//DD3       DD  DSN=DATEI3,DISP=(OLD,UNCATLG)
```

Der Eintrag für DATEI3 wird aus dem Katalog entfernt. Beim Entkatalogisieren einer Datei sollte SHR nicht benutzt werden, denn Dateien, zu denen von mehreren Jobs zugegriffen wird, dürfen nicht verändert werden. Wäre DATEI3 nicht katalogisiert, so könnte das System die Datei mit diesen Angaben nicht lokalisieren und würde deshalb den Step abbrechen.

```
//A         DD  DSN=DADATEI,DISP=(OLD,UNCATLG),UNIT=2311,
//             VOL=SER=DA0600
```

DADATEI wird aus dem Katalog gestrichen. Wäre DADATEI nicht katalogisiert, so würde UNCATLG wie KEEP wirken.

8.8.3 Abnormale Disposition

Die abnormalen Dispositionen werden nur wirksam, wenn der Step abnormal endet. Sie sind den normalen Dispositionen, PASS ausgenommen, gleich. KEEP, CATLG, UNCATLG und DELETE sind alle zugelassen: Z.B. DISP=(NEW,PASS,DELETE), DISP=(OLD,DELETE, CATLG), etc.

Wird keine abnormale Disposition angegeben und endet der Step dann abnormal, so wird die normale Disposition durchgeführt. Z.B. ist DISP=(OLD,KEEP) äquivalent mit DISP=(OLD,KEEP,KEEP). Die abnormale Disposition für temporäre Dateien ist stets DELETE, gleichgültig was kodiert wird.

Eine übergebene Datei (DISP=(..,PASS,..))führt bei abnormalem Stepende die abnormale Disposition aus, die angegeben wurde, als die Datei zuletzt übergeben worden war. War keine abnormale Disposition angegeben, so wird die Datei gelöscht, falls sie beim ersten Übergeben NEW war, andernfalls wird sie festgehalten (KEEP). Das folgende Beispiel veranschaulicht den Gebrauch der abnormalen Disposition.

```
//STEP1     EXEC PGM=ERZEUGE
//A         DD  DSN=JUNK,DISP=(,KEEP),UNIT=2314,VOL=SER=DA0200,
//             SPACE=(800,20)
```

Wenn STEP1 abnormal terminiert, wird die Datei JUNK festgehalten. Wenn man den Job, nachdem das Programm verbessert worden ist, mit

der gleichen JCL wieder abgibt, wird er wiederum abbrechen, dieses Mal aber weil die Datei JUNK auf dem Datenträger DA0200 bereits besteht. Man muß die Disposition in der DD-Karte A abändern in DISP=(OLD,KEEP). Diese Unbequemlichkeit wird vermieden, wenn man die Datei bei abnormaler Terminierung des Steps löscht.

```
//STEP1     EXEC PGM=ERZEUGE
//A         DD   DSN=JUNK,DISP=(,KEEP,DELETE),UNIT=2314,
//               VOL=SER=DA0200,SPACE=(800,20)
```

Endet STEP1 jetzt abnormal, so wird JUNK gelöscht, und die Disposition muß nicht in OLD abgeändert werden, wenn der Job wieder abgegeben werden soll. Wie das obige Beispiel zeigt, ist die abnormale Disposition besonders nützlich bei Plattendateien, weil nachfolgende Läufe von der Disposition der Dateien in den vorausgegangenen Läufen abhängig sein können. Die abnormale Disposition kann, über das Löschen unerwünschter Dateien hinaus, aber auch zum Aufbewahren der Datei benutzt werden.

```
//STEP1     EXEC PGM=EINS
//A         DD   DSN=DADAT,DISP=(,PASS),UNIT=2314,SPACE=(80,200)
```

Endet STEP1 abnormal, so wird DADAT gelöscht, weil bei einer Kodierung von PASS die übergebene Datei bei der abnormalen Beendigung des Steps ihre ursprüngliche Disposition wieder annimmt.

```
//STEP2     EXEC PGM=ZWEI
//B         DD   DSN=DADAT,DISP=(OLD,DELETE,CATLG)
```

Endet STEP2 abnormal, so wird DADAT katalogisiert und die Angaben UNIT und VOL werden im Katalog registriert. Wird die Restart-Einrichtung benutzt, um den Job mit STEP2 wieder anlaufen zu lassen, so braucht man die DD-Karte B nicht zu ändern.

Der Status, die normale Disposition und die abnormale Disposition einer Datei sind wahlfrei. Die folgenden Beispiele zeigen die wirksam werdenden Dispositionen, falls die Parameter nicht kodiert werden. Der Leser sollte die Beschreibung der Dispositionen noch einmal nachlesen, falls die angenommenen Dispositionen nicht klar werden.

```
//A         DD   UNIT=2314,SPACE=(80,100)
```

DISP=(NEW,DELETE,DELETE) wird unterstellt.

```
//B       DD  DISP=NEW,UNIT=2314,SPACE=(80,100)
```

DISP=(NEW, DELETE, DELETE) wird unterstellt.

```
//C       DD  DSN=LIB,DISP=SHR
```

DISP=(SHR, KEEP, KEEP) wird unterstellt.

```
//D       DD  DSN=DADAT,DISP=MOD,UNIT=2311,SPACE=(80,100)
```

DISP=(OLD, KEEP, KEEP) wird unterstellt, falls DADAT katalogisiert ist oder von einem vorangegangenen Step übergeben wurde.
DISP=(NEW, DELETE, DELETE) wird unterstellt, falls DADAT vorher nicht besteht.

```
//E       DD  DSN=DADAT,DISP=(,KEEP,DELETE),UNIT=2314,
//           SPACE=(80,20)
```

DISP=(NEW, KEEP, DELETE) wird unterstellt.

```
//F       DD  DSN=&&TEMP,DISP=(OLD,PASS,KEEP)
```

DISP=(OLD, PASS, DELETE) wird unterstellt.

```
//G       DD  DSN=DADAT,DISP=(OLD,,CATLG),UNIT=2311,
//           VOL=SER=DA0400
```

DISP=(OLD, KEEP, CATLG) wird unterstellt.

```
//H       DD  DSN=&&TEMP,DISP=(,KEEP),UNIT=2314,SPACE=(80,100)
```

DISP=(NEW, PASS, DELETE) wird unterstellt.

8.9 Parameter für den Datenträger: VOL

Mit dem Parameter VOL[3]) kann man
- einen bestimmten Datenträger,
- mehrere Datenträger,
- bestimmte Datenträger einer katalogisierten sich über mehrere Datenträger erstreckenden Datei und
- private Datenträger anfordern.

---

[3]) Der Parameter VOL kann auch als VOLUME kodiert werden. Z.B. sind VOL=SER=DA0200 und VOLUME=SER=DA0200 äquivalent.

Unter einem Datenträger (Volume) versteht man eine Magnetbandrolle, einen Plattenstapel, eine Unterzelle einer 2321 Datenzelle, eine Trommel oder den von einem Zugriffsmechanismus bedienten Teil eines IBM-2302-Plattenspeichers.

Jeder Datenträger kann mit einem Kennsatz versehen sein. Man unterscheidet zwei Kennsatztypen (eigentlich drei, wenn man das außen auf die Bandrolle oder den Plattenstapel aufgeklebte Etikett mitzählt):
- den Datenträgerkennsatz und
- den Datei-Kennsatz.

Beide, der Datenträgerkennsatz und der Datei-Kennsatz, sind auf dem Datenträger selbst gespeichert. Der Datenträgerkennsatz enthält die Datenträgernummer. Er wird vom System gelesen, um sicherzustellen, daß der erwartete Datenträger auch tatsächlich geladen ist.

Der Datenträgerkennsatz wird auch von den Routinen der AVR-Einrichtung (Automatic Volume Recognition) benötigt. Diese Einrichtung erlaubt es dem Operator, einen Datenträger zu laden, bevor er von einem Job verlangt wird. Die AVR-Routinen lesen den Datenträgerkennsatz und merken sich die Datenträgernummer. Wird der Datenträger verlangt, so weiß das System, daß dieser Datenträger bereits geladen ist, und die Verarbeitung kann sofort beginnen.

Datenträger können ständig (Permanently) geladen sein, vom Operator geladen werden (durch Operator Befehl) oder aber geladen werden, weil sie vom Job eines Benutzers verlangt werden. Datenträger sind ständig geladen (*Permanently/Resident*), falls sie wichtige Teile der System-Residenz enthalten, falls der Datenträger physisch nicht entladen werden kann (wie etwa eine Trommel) oder falls sie in der Installation den Status "ständig geladen" (Permanently/Resident) durch einen Eintrag in dem besonderen Glied PRESRES der Datei SYS1.PARMLIB erhalten haben. Plattenstapel erhalten relativ häufig den Status "ständig geladen", obwohl 2311 und 2314 Plattenstapel natürlich vom Operator und durch Anforderung des Benutzers geladen werden können. Bänder dagegen werden gewöhnlich erst geladen, wenn sie vom Job des Anwenders verlangt worden sind.

Der Parameter VOL muß für eine bestehende Datei stets die Datenträgernummer angeben, es sei denn, die Datei ist katalogisiert oder sie ist übergeben worden (PASS). Eine neue Datei kann auf bestimmten oder beliebigen (*Specific* und *Nonspecific*) Datenträgern eingerichtet werden. Der VOL-Parameter fordert bestimmte Datenträger an. Läßt man ihn

aber weg, so sucht das System selbst die passenden Datenträger für den angegebenen Geräte-Typ aus. Die Installation kann durch einen PRESRES Eintrag solche passenden Datenträger bestimmen, die dann bei der Anforderung nach einem beliebigen (nonspecific) Datenträger ausgewählt werden. Ein solcher Eintrag kann einen Datenträger zum Public- (nimmt nur temporäre Dateien auf), Storage- (nimmt temporäre und nicht-temporäre Dateien auf), Private (ist für alle Anforderungen nach beliebigen Datenträgern gesperrt) oder Scratch- (Scratch ist nur für Bänder möglich) Datenträger erklären.

8.9.1 Anforderung für einen bestimmten[4]) Datenträger: SER, REF

Um einen bestimmten Datenträger anzufordern, wird VOL=SER=*datenträgernummer* für einen Datenträger bzw. VOL=SER=(*datenträgernummer*, . . , *datenträgernummer*) für mehrere Datenträger kodiert. Die Datenträgernummern enthalten bis zu 6 alphanumerische (A-Z, 0-9) oder nationale (@$#) Zeichen, sowie den Bindestrich (-). Sonderzeichen (Leerstelle, . /'()*&+=) können ebenfalls verwendet werden, wenn man die Datenträgernummer in Hochkommas einschließt. Ein Hochkomma in der Datenträgernummer muß durch zwei aufeinanderfolgende Hochkommas dargestellt werden.

```
//A        DD  DSN=DATEI1,DISP=(,CATLG),UNIT=2311,
//            VOL=SER=PACK12,SPACE=(80,200)
```

Ein 2311 Plattenstapel mit der Datenträgernummer PACK12 wird angefordert. Ist der Plattenstapel nicht geladen, so wird das System den Operator auffordern, ihn zu laden.

```
//B        DD  DSN=DATEI2,DISP=(OLD,KEEP),UNIT=(BAND,3),
//            VOL=SER=(BD1,BD2,BD3)
```

Es werden die drei Bandrollen BD1, BD2, BD3 zum parallelen Laden angefordert. Eine DD-Karte, die mehrere Datenträgernummern angibt, kann im Parameter VOL=SER unterbrochen und auf einer Folgekarte fortgesetzt werden, wenn die Unterbrechung nach einer vollständigen Datenträgernummer, das Komma eingeschlossen, erfolgt.

---

[4]) Anmerkung des Übersetzers: In der englischen Literatur spricht man von einem Specific Request oder kürzer von einem Specific Volume.

Anstatt bestimmte Datenträgernummern zu kodieren, kann man auch die gleichen Datenträger anfordern, die von einer anderen Datei benutzt werden. Wenn die andere Datei katalogisiert ist oder von einem vorausgegangenen Step übergeben wurde, so kodiert man VOL=REF=name, wobei name der Name der anderen Datei ist. Der Name darf keine Sonderzeichen enthalten. Ist die Datei aber nicht katalogisiert oder wurde sie nicht übergeben (PASS), bzw. hat sie einen temporären Namen erhalten, so muß man VOL=REF=*.rückbezug kodieren. Der Rückbezug zielt auf die vorangegangene DD-Karte, die die andere Datei beschreibt. Sollte sich die bezogene Datei über mehrere Datenträger erstrecken, so wird für Bänder nur die letzte Bandrolle angesprochen, bei Plattendateien über mehrere Datenträger dagegen alle Datenträger. Zielt der Rückbezug auf eine DD-Karte, die eine Pseudodatei beschreibt (DUMMY,NULLFILE), so bekommt die DD-Karte, die diesen Rückbezug benutzt, den gleichen Status.

```
//STEP1     EXEC PGM=ANFANG
//A         DD   DSN=DATEI1,DISP=(,PASS),UNIT=2314,VOL=SER=DA0200,
//              SPACE=(800,100)
//B         DD   DSN=DATEI2,DISP=(,PASS),VOL=REF=*.A,SPACE=(200,50)
```

Der Datenträger DA0200 wird ebenfalls der Datei DATEI2 zugeordnet. Der UNIT-Parameter ist nicht notwendig, denn das System weiß bereits, welchem Gerät dieser Plattenstapel zugeordnet ist.

```
//STEP2     EXEC PGM=ENDE
//C         DD   DSN=DATEI3,DISP=(,CATLG),VOL=REF=*.STEP1.A,
//              SPACE=(400,30)
                   ODER
//C         DD   DSN=DATEI3,DISP=(,CATLG),VOL=REF=DATEI1,
//              SPACE=(400,30)
```

Auch die Datei DATEI3 wird dem Plattenstapel DA0200 zugeordnet.

8.9.2 Besondere Optionen

Der VOL-Parameter stellt besondere Optionen zur Verfügung
- um einen Privaten Datenträger anzufordern,

- um sicherzustellen, daß Private Datenträger zwischen den Steps geladen bleiben,
- um mehrere Datenträger anzufordern und
- um einen bestimmten Datenträger einer Datei über mehrere Datenträger anzufordern.

Alle Unterparameter sind wahlfrei und können in jeder beliebigen Kombination kodiert werden. Bei Weglassung eines Positionsunterparameters muß ein Auslassungskomma kodiert werden.

```
                                                SER=..
                                                REF=..
VOL=(PRIVATE,RETAIN, folgenummer, anzahl,-------)
```

8.9.2.1 *Datenträger vom Typ Private: PRIVATE*

Kodiert man VOL=(PRIVATE,..), so bekommt die Datei während des Steps die alleinige Verfügung über den ladbaren Datenträger. Auf diesen Datenträgern werden keine anderen Dateien eingerichtet; nach seiner Benutzung wird er mit Stepende entladen. Die PRIVATE-Anforderung kann gezielt oder nicht gezielt sein[5]); ist sie nicht gezielt, fehlt also die Angabe einer Datenträgernummer, so fordert das System den Operator auf, einen Datenträger vom Typ PRIVATE auszuwählen. Richtet sich die Anforderung auf einen ständig geladenen Datenträger (Permanently/Resident), so wird der Datenträger zugeordnet, PRIVATE aber ignoriert.

Für Banddateien wird PRIVATE unterstellt, wenn sie nicht-temporär sind oder wenn für sie ein bestimmter Datenträger angefordert wird. Ist das nicht der Fall, so heißt der Datenträger Scratch-Datenträger. Bei Stepende kann er entladen werden, um von einem anderen Job verwendet zu werden. Wird ein Scratch-Magnetband an einen nachfolgenden Step übergeben (PASS), so sollte in dem aufnehmenden Step PRIVATE kodiert werden, damit die Benutzung durch einen anderen Job verhindert wird.

```
//STEP1    EXEC PGM=PROGM1
//A        DD  DSN=DATEI,DISP=(,CATLG),UNIT=2311,
//             VOL=(PRIVATE,SER=DA0200),SPACE=(80,100)
```

[5]) Anmerkung des Übersetzers: Specific or Nonspecific Request, gemeint ist die Anforderung eines Datenträgers unter Angabe einer Datenträgernummer bzw. ohne Angabe einer solchen.

Das System wird den Operator auffordern den Plattenstapel DA0200 zu laden. Diesem Plattenstapel werden keine anderen Dateien zugeordnet. Bei Stepende wird er nach seiner Benutzung entladen.

```
//B          DD   UNIT=BAND,DISP=(,PASS)
```

Es wird ein Scratch-Magnetband verlangt.

```
//STEP2      EXEC PGM=PROGM2
//C          DD   DSN=*.STEP1.B,DISP=(OLD,DELETE),VOL=PRIVATE
```

Das Scratch-Band des vorausgegangenen Steps wird benutzt.

8.9.2.2 *Halten von Private Datenträgern: RETAIN*

Um einen Datenträger auch zwischen den Steps auf der Einheit zu halten, kodiert man VOL=(PRIVATE,RETAIN,..). Der Datenträger bleibt geladen, bis er in einem nachfolgenden Step wieder benutzt wird oder bis das Ende des Jobs erreicht ist. Das Geladenbleiben bezieht sich bei einer Datei über mehrere Datenträger natürlich nur auf den letzten. Bei Bändern wird RETAIN oft benutzt, wenn diese in mehreren Steps benutzt werden. Es verringert die Rüstzeiten für die Bänder. (Auch bei der Disposition PASS bleiben die Datenträger zwischen den Steps geladen.)

Wird RETAIN nicht kodiert und wurde die Datei nicht übergeben (PASS), so wird ein ladbarer Datenträger bei Stepende entladen. Anschließend kann es vorkommen, daß der Operator den gleichen Datenträger auf einem anderen E/A-Gerät laden muß, falls ein nachfolgender Step ihn verlangt. (Das Makro CLOSE der Assembler-Sprache hat vor den Parametern der DD-Karte Vorrang und kann eine andere Disposition enthalten.)

```
//A          DD   DSN=PLATEI,DISP=(,CATLG),UNIT=PLATTEN,
//                VOL=(PRIVATE,RETAIN,SER=DA0300),
//                SPACE=(1600,20)
```

Der Plattenstapel DA0300 wird bei Stepende nicht entladen.

8.9.3 Folgenummer

Wird eine Datei über mehrere Datenträger katalogisiert, so werden alle Datenträgernummern im Katalog festgehalten. Um die Verarbeitung mit

einem anderen als dem ersten Datenträger zu beginnen, muß man VOL=(...,...,*folgenummer*,...) kodieren. Die Folgenummer ist 1- bis 4-stellig. Wird die Folgenummer weggelassen oder wurde null kodiert, so beginnt die Verarbeitung mit dem ersten Datenträger.

```
//A        DD  DSN=MEHRVOL,DISP=OLD,VOL=(PRIVATE,RETAIN,2)
```

Die Verarbeitung beginnt mit dem 2. Datenträger der Datei MEHRVOL. Die Angaben für die Datenträgernummern und das E/A-Gerät wurden weggelassen, weil sie im Katalog enthalten sind.

```
//B        DD  DSN=IRS,DISP=OLD,VOL=(,,3)
```

Die Verarbeitung beginnt mit dem 3. Datenträger der katalogisierten Datei IRS.

8.9.4 Anzahl

Wenn sich eine Datei über mehrere Datenträger erstreckt, kann man die Datenträger entweder gezielt anfordern, indem man die Datenträgernummern im Parameter SER anführt, oder man gibt nur die Anzahl der Datenträger für eine nicht gezielte Anforderung an. Für eine nicht gezielte Datenträgeranforderung kodiert man VOL=(...,...,...,*anzahl*,...), wobei anzahl die Zahl (2-255) der benötigten Datenträger angibt. Die Anzahl kann auch bei einer katalogisierten oder übergebenen (PASS) Datei kodiert werden, um anzuzeigen, daß möglicherweise zusätzliche Datenträger erforderlich sind. Wird die Anzahl weggelassen oder kodiert man null, so wird ein Datenträger unterstellt.

```
//A        DD  DSN=IRS,DISP=(,CATLG),UNIT=BAND,VOL=(PRIVATE,,,2)
```

Der Operator wird aufgefordert, maximal 2 Bandrollen zuzuordnen und zu laden.

Wird eine Datei über mehrere Datenträger erzeugt oder erweitert, und ist dabei die für die Datei erforderliche Datenträgeranzahl unbekannt, so kann die maximale Anzahl kodiert werden.

```
//A        DD  DSN=DATEI,DISP=(,CATLG),UNIT=BAND,
//            VOL=(,,,6,SER=(BD0100,BD0200))
```

Die Bandrollen BD0100 und BD0200 werden der Datei zugeordnet. Der Operator kann aber aufgefordert werden, bis zu 4 zusätzliche Datenträger zuzuordnen und zu laden, falls das nötig sein sollte.

Die Datenträgernummern aller benutzten Datenträger werden im Katalog festgehalten.

Als Alternative kann verzögertes Laden kodiert werden.

```
//A          DD   DSN=DATEI,DISP=(,CATLG),UNIT=(BAND,,DEFER),
//                VOL=SER=(BD0100,BD0200)
```

Die Wirkung dieser Anweisung ist die gleiche, wie die der vorausgegangenen Anweisung, die Tatsache ausgenommen, daß die Datei hinsichtlich der Datenträgeranzahl, auf die sie sich ausdehnen kann, nicht beschränkt ist.

Übersteigt die Datenträgeranzahl die mit dem UNIT-Parameter angeforderte Zahl von Plattenspeichereinheiten, so muß PRIVATE ebenfalls kodiert werden, also beispielsweise: UNIT=(2314,2),VOL=(PRIVATE,,, 3). Wird PRIVATE nicht kodiert, und verlangt der Job trotzdem mehr als die kodierte Einheitenanzahl, so wird der Job abnormal beendet. Das Kodieren von PRIVATE ist für Bandeinheiten nicht nötig.

Die Anzahl der Datenträger für eine einzelne DD-Karte ist auf 255 beschränkt, die eines Jobsteps auf 4095. Bei PCP-Systemen sollte SYSIN nicht als Datenträgernummer verwendet werden, weil PCP den Namen für eigene Zwecke verwendet. Jeder Datenträger in einer Installation sollte, unabhängig vom Datenträgertyp, seine eigene unverwechselbare Datenträgernummer haben.

Es sollen nun zwei besondere Probleme behandelt werden. Angenommen, man möchte eine Datei auf einen Datenträger schreiben, dessen Datenträgernummer aber unbekannt ist. Um das zu erreichen, muß folgendes in der DD-Karte kodiert werden.

```
//          DD   DISP=(NEW,...),VOL=PRIVATE,UNIT=(ADRESSE,,DEFER
```

Beispeilsweise könnte man sich vorstellen, daß der Aufkleber einer Bandrolle verloren gegangen ist, und man deshalb die Datenträgernummer nicht kennt. Der folgende Job eröffnet eine Datei für Ausgabe und katalogisiert dann das Band. Dadurch wird die Datenträgernummer in der Terminierungsmeldung des Systems angedruckt und außerdem im Katalog festgehalten.

```
//STEP1     EXEC PGM=OPEN
```

Das Programm muß die Datei eröffnen.

```
//A        DD  DSN=WELCHE,DISP=(,CATLG),VOL=PRIVATE,
//            UNIT=(ODO,,DEFER)
```

Das zweite Problem besteht darin, daß eine Datei im gleichen Step zweimal benutzt wird. Sie könnte vielleicht zuerst als Ausgabedatei eröffnet werden, wieder geschlossen werden, um dann als Eingabedatei noch einmal eröffnet zu werden. Normalerweise reicht dafür eine DD-Karte aus. Das Programm könnte aber auch so geschrieben sein, daß zwei DD-Karten nötig sind. Die zweite DD-Karte muß folgende Angaben enthalten:

DSN=*.ddname,VOL=REF=*.ddname,DISP=(OLD,...)

Das folgende Beispiel zeigt ein Programm, das eine Datei in einem Step zweimal benutzt.

```
//STEP2    EXEC PGM=EINAUS
//A        DD  UNIT=2314,SPACE=(80,200)
```

Die DD-Karte erzeugt eine Scratch-Datei.

```
//B        DD  DSN=*.A,VOL=REF=*.A,DISP=OLD
```

Die DD-Karte B bezieht sich auf die Scratch-Datei.

8.10 Zusammenfassung

Eine DD-Karte muß für jede im Step verwendete Datei nach der EXEC-Karte eingefügt werden. Die allgemeine Form der DD-Karte ist:

```
                DATA
                DUMMY
//ddname    DD  -------, schlüsselwortparameter
```

In diesem Kapitel wurden die im folgenden aufgeführten Parameter besprochen:

Der DCB-Parameter

DCB=(*unterparameter,...,unterparameter*)
DCB=(*name,unterparameter,...,unterparameter*)
DCB=(*.rückbezug,unterparameter,...,unterparameter*)

Der DSN-Parameter

DSN=*name* DSN=&&*name*
DSN=*.rückbezug
DSN=*name* (*glied*) DSN=&&*name* (*glied*)
DSN=*name* (*zahl*)
DSN=*name* (INDEX) DSN=&&*name* (INDEX)
DSN=*name* (PRIME) DSN=&&*name* (PRIME)
DSN=*name* (OVFLOW) DSN=&&*name* (OVFLOW)

Der UNIT-Parameter

```
          typ
          gruppe  P
          adresse  volumes
UNIT=(------,------ ,DEFER,SEP=(ddname,...,ddname)
UNIT=AFF=ddname
```

Der DISP-Parameter

```
DISP=(status, normale disposition, abnormale disposition)
                KEEP
          NEW   DELETE    KEEP
          MOD   PASS      DELETE
          OLD   CATLG     CATLG
          SHR   UNCATLG   UNCATLG
DISP=(----,-----------,----------)
```

Der VOL-Parameter

```
VOL=(PRIVATE,RETAIN, folgenummer, anzahl,
          REF=name
          REF=*.rückbezug
          SER=(datenträgernummer,...,datenträgernummer)
          ------------------------------------------)
```

9. Katalogisierte Verfahren

Dieses Kapitel bringt die Diskussion vielfältiger Einzelheiten der Parameter der DD-Karte. Es werden verknüpfte Dateien, die Plazierung von DD-Karten und katalogisierte Verfahren beschrieben.

9.1 Verknüpfung von Dateien

Es ist möglich, mehrere getrennte Eingabe-Dateien einzulesen, als ob es sich um eine einzige Datei handelt, indem man diese Dateien miteinander *verknüpft* (Concatenation). Die Dateien können auf verschiedenen Ausgabegeräten liegen, jedoch müssen sie in ihren Charakteristiken übereinstimmen (Satzformat, Satzlänge). Dateien werden miteinander verknüpft, indem man eine DD-Karte für jede Datei kodiert, und diese so hintereinander legt, wie die Dateien gelesen werden sollen. Der Parameter ddname wird nur auf der ersten Karte kodiert. Die maximale Zahl der verknüpfbaren sequentiellen Dateien beträgt 255. Im folgenden Beispiel werden die Dateien DATEI1, DATEI2, DATEI3 und die Datenkarten im Eingabestrom sequentiell als eine Datei eingelesen.

```
//EINDD     DD  DSN=DATEI1,DISP=SHR
//          DD  DSN=DATEI2,DISP=SHR,UNIT=2311,VOL=SER=DA0222
//          DD  DSN=DATEI3,DISP=OLD,UNIT=BAND,VOL=SER=BD0002
//          DD  *
  ***************
  * DATENKARTEN *
  ***************
/*
```

Untergliederte Dateien, maximal 16, lassen sich ebenfalls miteinander verknüpfen, aber sie sind nicht mit sequentiellen Dateien verknüpfbar. Da jedoch einzelne Glieder von untergliederten Dateien wie sequentielle Dateien behandelt werden, kann man Glieder wiederum miteinander und mit anderen sequentiellen Dateien verknüpfen. Im folgenden Beispiel wird eine drei Glieder enthaltende untergliederte Datei sequentiell durch Verknüpfung der Glieder eingelesen.

```
//EINDD     DD  DSN=LIB(EINS),DISP=SHR
//          DD  DSN=LIB(ZWEI),DISP=SHR
//          DD  DSN=LIB(DREI),DISP=SHR
```

Die Glieder EINS, ZWEI und DREI der Bibliothek LIB werden in genau dieser Reihenfolge eingelesen. Das nächste Beispiel verknüpft sequentielle Dateien mit dem Glied einer untergliederten Datei.

```
//EINDD     DD  DSN=LIB(EINS),DISP=SHR
//          DD  *
  ***************
  * DATENKARTEN *
  ***************
/*
//          DD  DSN=DATEI1,DISP=SHR
```

Zuerst wird das Glied EINS der Bibliothek LIB eingelesen, anschließend folgen die Datenkarten und zum Schluß wird die sequentielle Datei DATEI1 gelesen. Wird eine untergliederte Datei mit einer sequentiellen Datei verknüpft, so darf der Name des Gliedes bei der untergliederten Datei nicht fehlen. Im folgenden Beispiel geschieht beim Lesen von EINDD ein E/A-Fehler, weil die Datei LIB eine untergliederte Datei ist. Um die Datei LIB als ganzes einlesen zu können, müßte man alle Glieder miteinander verknüpfen.

```
//EINDD     DD  DSN=DATEI1,DISP=SHR
//          DD  DSN=LIB,DISP=SHR
```

9.2 Plazierung von DD-Karten

Alle DD-Karten für einen Step müssen der EXEC-Karte unmittelbar folgen.

```
//LAUF      PROC
//STEP1     EXEC PGM=PROGM1
//EINGABE   DD  DSN=DATEI1,DISP=OLD
//SCRATCH   DD  DSN=&&TEMP,UNIT=2311,VOL=(PRIVATE,SER=DA0200),
//              SPACE=(TRK,200),DCB=(RECFM=FB,BLKSIZE=160,
//              LRECL=80)
```

EINGABE und SCRATCH gehören zum STEP1.

```
//STEP2     EXEC PGM=PROGM2
//AUSGABE   DD  DUMMY,SYSOUT=A
```

AUSGABE gehört zu STEP2.

Die Parameter der DD-Karten in katalogisierten Verfahren können überschrieben oder gelöscht werden. Neue Parameter sowie ganze DD-Karten können hinzugefügt werden. Der ddname einer zu überschreibenden oder hinzuzufügenden DD-Karte eines Verfahrens wird um den Stepnamen verlängert. Stimmt der ddname der veränderten Karte mit dem ddname einer Karte des Verfahrens überein, so werden die Parameter dieser Karte überschrieben. Stimmen die verglichenen ddname nicht überein, so wird die verändernde Karte hinzugefügt. Die DD-Karten müssen in der Reihenfolge, in der sie im Verfahren liegen, überschrieben werden. Hinzuzufügende Karten müssen den überschreibenden Karten folgen.

Hinzuzufügende DD-Karten werden als Ganzes hinzugefügt. Eine überschriebene DD-Karte dagegen wird Parameter für Parameter modifiziert. Ein Parameter wird überschrieben, wenn der Parameter auf der überschreibenden Karte mit einem Parameter der Verfahrenskarte übereinstimmt. Stimmt er nicht überein, so wird dieser Parameter aus der überschreibenden Karte der Verfahrenskarte hinzugefügt. Die Reihenfolge der Parameter in der überschreibenden DD-Karte spielt keine Rolle. Die in dem Verfahren durch die überschreibenden Karten erzielten Veränderungen sind nur für den einen Lauf wirksam; sie erreichen keine ständige Veränderung des katalogisierten Verfahrens. Das folgende Beispiel veranschaulicht die Veränderung von DD-Karten im oben angeführten Verfahren LAUF.

```
//AUFRUF    EXEC LAUF
//STEP1.EINGABE DD  DSN=DATEI5,UNIT=2311,VOL=SER=DA0222
```

EINGABE stimmt mit einem ddname in STEP1 überein, daher werden Parameter überschrieben bzw. hinzugefügt. Der Parameter DSN wird überschrieben, DISP bleibt unverändert, UNIT und VOL werden hinzugefügt.

```
//STEP1.SCRATCH DD  UNIT=2314
```

Der Parameter UNIT wird überschrieben. Die Karte zum Überschreiben von SCRATCH muß der Karte folgen, die EINGABE überschreiben soll; denn SCRATCH liegt auch im Verfahren hinter EINGABE.

```
//STEP2.AUSGABE DD  SYSOUT=B
```

Der Parameter SYSOUT wird überschrieben.

```
//STEP2.WEITDD DD  DSN=TROMMEL,DISP=SHR
```

Weil diese DD-Karte hinzugefügt werden soll, muß sie der AUSGABE überschreibenden Karte folgen.

Für diesen einen Lauf wird das katalogisierte Verfahren wie folgt interpretiert.

```
//STEP1    EXEC PGM=PROGM1
//EINGABE  DD  DSN=DATEI5,DISP=OLD,UNIT=2311,VOL=SER=DA0222
//SCRATCH  DD  DSN=&&TEMP,UNIT=2314,VOL=(PRIVATE,SER=DA0200),
//             SPACE=(TRK,200),DCB=(RECFM=FB,BLKSIZE=160,
//             LRECL=80)
//STEP2    EXEC PGM=PROGM2
//AUSGABE  DD  DUMMY,SYSOUT=B
//WEITDD   DD  DSN=TROMMEL,DISP=SHR
```

Die vom System auf Grund der Angabe MSGLEVEL=1 in der JOB-Karte erzeugte Liste enthält die Karten wie sie kodiert wurden, und nicht wie sie das System interpretiert. Jede überschreibende DD-Karte wird unmittelbar vor der Karte, die sie überschreibt, angedruckt; hinzugefügte DD-Karten folgen am Ende des Steps. Die im Eingabestrom enthaltenen DD-Karten werden mit // in den Spalten 1 und 2, die Karten des Verfahrens mit XX und die überschriebenen Karten des Verfahrens mit X/ angedruckt. Die vom System auf Grund der Angabe MSGLEVEL=1 erzeugten Listen des obigen Beispiels würden wie folgt aussehen:

```
//AUFRUF    EXEC LAUF
XXSTEP1     EXEC PGM=PROGM1
//STEP1.EINGABE DD  DSN=DATEI5,UNIT=2311,VOL=SER=DA0222
X/EINGABE   DD  DSN=DATEI1,DISP=OLD
                .
                .
                .
```

Unglücklicherweise wird die Art, in der das System die Karten interpretiert, nicht gezeigt, und man sollte deshalb sorgfältig prüfen, ob die überschreibenden und hinzugefügten DD-Karten in der richtigen Reihenfolge sind. Das folgende Beispiel zeigt einen durch vertauschte DD-Karten hervorgerufenen häufigen Fehler.

```
//AUFRUF    EXEC LAUF
//STEP2.WEITDD DD  DSN=TROMMEL,DISP=SHR
//STEP2.AUSGABE DD  DSN=DATEI3,DISP=OLD
```

WEITDD sollte nach AUSGABE folgen, denn es ist ja eine hinzuzufügende DD-Karte. Das System entdeckt diesen Fehler allerdings nicht und nimmt an, AUSGABE solle auch hinzugefügt werden. Da nun zwei DD-Karten im Step den gleichen ddname haben, wird die zweite AUSGABE DD-Karte ignoriert. Dadurch wird also eine DD-Karte, die zum Überschreiben gedacht war, ignoriert, ohne daß eine Fehlermeldung angedruckt wird. Dieser Fehler ist besonders schwer zu entdecken, wenn MSGLEVEL=1 in der JOB-Karte zum Andrucken der JCL nicht kodiert wird. STEP2 sieht für diesen Lauf wie folgt aus:

```
XXSTEP2     EXEC PGM=PROGM2
XXAUSGABE   DD  DUMMY,SYSOUT=A
//STEP2.WEITDD DD  DSN=TROMMEL,DISP=SHR
//STEP2.AUSGABE DD  DSN=DATEI3,DISP=OLD
```

In der überschreibenden DD-Karte können die Parameter in jeder beliebigen Reihenfolge kodiert werden; diejenigen, die im Verfahren vorkommen, werden überschrieben, diejenigen, die nicht vorkommen, werden hinzugefügt. Alle Parameter, der DCB-Parameter ausgenommen, werden als Ganzes überschrieben. Statt des DCB-Parameters müssen die einzelnen DCB-Unterparameter überschrieben oder hinzugefügt werden.

```
//AUFRUF    EXEC LAUF
//STEP1.SCRATCH DD  DCB=BLKSIZE=800,VOL=SER=DA0666
```

Die SCRATCH DD-Karte wird interpretiert als:

```
//SCRATCH   DD  DSN=&&TEMP,UNIT=2311,VOL=SER=DA0666,
//              SPACE=(TRK,200),DCB=(RECFM=FB,BLKSIZE=800,
//              LRECL=80)
```

Parameter werden durch das Kodieren des Schlüsselwortes, gefolgt von einem Gleichheitszeichen und unter Weglassung einer Schlüsselwortangabe gelöscht. Um den ganzen DCB-Parameter zu löschen, muß jeder einzelne DCB-Unterparameter gelöscht werden. (DCB= kann nicht zur Löschung des ganzen DCB-Parameters kodiert werden.)
Um eine Pseudodatei (DUMMY oder DSN=NULLFILE) zu löschen, kodiert man den Parameter DSN in der überschreibenden Karte. Sollte in der DD-Karte der Dateiname nicht notwendig sein, kodiert man einfach DSN=.

```
//AUFRUF    EXEC LAUF
//STEP1.SCRATCH DD  DSN=,DCB=(RECFM=F,BLKSIZE=)
//STEP2.AUSGABE DD  DSN=
```

SCRATCH und AUSGABE werden interpretiert als:

```
//SCRATCH   DD  UNIT=2311,VOL=(PRIVATE,SER=DA0200),
//              SPACE=(TRK,200),DCB=(RECFM=F,LRECL=80)
//AUSGABE   DD  SYSOUT=A
```

Bringt man durch den Überschreibungsvorgang in eine DD-Karte Parameter, die andere in ihr enthaltene Parameter ausschließen (z.B. SYSOUT und DISP), so werden die so ausgeschlossenen Parameter automatisch gelöscht. Diese dürfen auf keinen Fall in der überschriebenen DD-Karte extra gelöscht werden.

Verknüpfte DD-Karten innerhalb eines katalogisierten Verfahrens müssen einzeln in der gleichen Reihenfolge überschrieben werden, in der sie im Verfahren auftreten. Das folgende Verfahren enthält eine verknüpfte Datei.

```
//LAUFA    PROC
//STEP1    EXEC PGM=PROGM1
//WORK     DD  DSN=LIBA,DISP=SHR
//        DD  DSN=LIBB,DISP=SHR
```

Soll eine bestimmte DD-Karte der verknüpften Datei unverändert bleiben, so muß man das Operandenfeld in der überschreibenden DD-Karte leer lassen.

```
//AUFRUF   EXEC LAUFA
//STEP1.WORK DD  DSN=LIBRARY
//        DD
//        DD  DSN=LIBC,DISP=SHR
```

Die erste DD-Karte wird überschrieben, die zweite bleibt unverändert und eine dritte Datei wird mit den vorausgegangenen verknüpft. Die DD-Karte WORK wird wie folgt interpretiert:

```
//WORK     DD  DSN=LIBRARY,DISP=SHR
//        DD  DSN=LIBB,DISP=SHR
//        DD  DSN=LIBC,DISP=SHR
```

Ein dem ddname angefügter Stepname hat keine Auswirkung, falls kein katalogisiertes Verfahren aufgerufen wird. Die folgenden beiden steps werden in der gleichen Weise behandelt.

```
//STEP1    EXEC PGM=EINS
//EIN      DD  DSN=LIB,DISP=SHR
                ODER
//STEP1    EXEC PGM=EINS
//GO.EIN   DD  DSN=LIB,DISP=SHR
```

9.3 Katalogisierte Verfahren

Die katalogisierten Verfahren vereinfachen die JCL erheblich; alle häufig benutzten Verfahren sollten daher katalogisiert werden. Ein katalo-

gisiertes Verfahren ist ein Satz von JCL-Anweisungen, die als Glied in einer untergliederten Datei auf einem Plattenspeicher stehen. Katalogisierte Verfahren sind normalerweise in der System Bibliothek SYS1.PROCLIB enthalten. In MFT- und MVT-Systemen können andere Bibliotheken in der Reader-Prozedur mit der SYS1.PROCLIB verknüpft werden. Das ist für das Testen von Verfahren sehr bequem. Da Bibliotheken in der Reihenfolge ihrer Verknüpfung durchsucht werden, könnte man es so einrichten, daß eine erste Bibliothek voll ausgetestete Verfahren enthält, eine zweite eigene Anwenderverfahren und eine dritte Bibliothek endlich zu testende Verfahren.

9.3.1 Hinzufügung von Verfahren zur Verfahrensbibliothek

Jedes katalogisierte Verfahren muß einen unverwechselbaren 1-8 Zeichen langen Namen erhalten. Er darf beliebige alphanumerische (A-Z, 0-9) oder nationale (@$#) Zeichen enthalten. Das IBM-Dienstprogramm IEBUPDTE dient zum Hinzufügen und Ersetzen von Verfahren. Das folgende Beispiel zeigt seine Anwendung.

```
//UPDTE     EXEC PGM=IEBUPDTE,REGION=24K
```

Das Programm IEBUPDTE ist in der SYS1.LINKLIB enthalten. Der Parameter REGION ist nur für MVT-Systeme nötig.

```
//SYSPRINT DD  SYSOUT=A
```

SYSPRINT definiert eine Druckdatei für das Protokoll der IEBUPDTE-Nachrichten.

```
//SYSUT1   DD  DSN=SYS1.PROCLIB,DISP=OLD
```

SYSUT1 definiert die Eingabedatei. Sollen die Verfahren zu einer anderen Bibliothek hinzugefügt werden, so muß SYS1.PROCLIB durch den Namen der Bibliothek ersetzt werden.

```
//SYSUT2   DD  DSN=*.SYSUT1,DISP=OLD
```

SYSUT2 beschreibt die Ausgabedatei - die gleiche, die für die Eingabe benutzt wird.

```
//SYSIN    DD  DATA
```

DATA, das im nächsten Kapitel beschrieben wird, sollte in der SYSIN Karte stets statt * benutzt werden, wenn die Datenkarten JCL mit // in den Spalten 1 und 2 enthalten.

```
   ADD
   REPL
./ ----     NAME=VERFAHRENSNAME,LIST=ALL
   ****************************
   * JCL-KARTEN DES VERFAHRENS *
   ****************************
/*
```

Um ein neues Verfahren hinzuzufügen, kodiert man ADD, dagegen REPL, wenn ein bestehendes Verfahren ersetzt werden soll. LIST=ALL bewirkt den Andruck aller Karten des Verfahrens in der Datei SYSPRINT. Der Wert Verfahrensname ist der Name des hinzuzufügenden oder zu ersetzenden Verfahrens. Man kann jede beliebige Anzahl von ADD- und REPL-Karten, gefolgt von der JCL des Verfahrens, einfügen.
Ein katalogisiertes Verfahren kann aus vielen Jobsteps bestehen. Verfahren dürfen alle JCL-Anweisungen außer den folgenden enthalten:

1) EXEC-Karten, die katalogisierte Verfahren aufrufen. (Ein katalogisiertes Verfahren darf kein anderes katalogisiertes Verfahren aufrufen.)
2) JOB-, Begrenzungs- (/*) oder Null- (//) Karten.
3) JOBLIB DD-Karten.
4) DD * oder DD DATA Karten.

Wir sind jetzt in der Lage, ein Verfahren in eine Bibliothek einzufügen. Ein offensichtlich sehr notwendiges Verfahren ist das Verfahren, das seinerseits Verfahren hinzufügt. Der passende Name für ein solches Verfahren ist ADDPROC.

```
//TEST     JOB 1776,MUELLER
//UPDTE    EXEC PGM=IEBUPDTE,REGION=24
//SYSPRINT DD  SYSOUT=A
//SYSUT1   DD  DSN=SYS1.PROCLIB,DISP=OLD
//SYSUT2   DD  DSN=*.SYSUT1,DISP=OLD
//SYSIN    DD  DATA
./ ADD          NAME=ADDPROC,LIST=ALL
//ADDPROC  PROC
//GO       EXEC PGM=IEBUPDTE,REGION=24
//SYSPRINT DD  SYSOUT=A
//SYSUT1   DD  DSN=SYS1.PROCLIB,DISP=OLD
//SYSUT2   DD  DSN=*.SYSUT1,DISP=OLD
/*
```

Um jetzt ein Verfahren hinzuzufügen, braucht man nur den folgenden Step auszuführen.

```
//ADD       EXEC ADDPROC
//GO.SYSIN DD  DATA
  ************************************************************
  * ADD- UND REPL-KARTEN MIT DEN NOTWENDIGEN JCL-KARTEN *
  ************************************************************
/*
```

Manchmal hat man ein Programm, das Karten einliest, die sich selten von Lauf zu Lauf verändern. Das nachfolgende Beispiel zeigt ein solches Programm.

```
//GO        EXEC PGM=EINS,REGION=88K
//SYSIN     DD  *
  ***************
  * DATENKARTEN *
  ***************
/*
```

Man könnte diesen Step als Verfahren aufnehmen wollen und die sich selten ändernden Datenkarten ebenso. Die Verfahrensbibliothek enthält Karten. Man kann alle Datenkarten, nicht nur JCL-Karten, als Glied hinzufügen. Die DD-Karte SYSIN muß dann auf das Glied verweisen, das diese Datenkarten enthält. Der folgende Step fügt das Verfahren und die Datenkarten zur SYS1.PROCLIB hinzu.

```
//ADD       EXEC ADDPROC
//GO.SYSIN DD  DATA
./ ADD      NAME=LAUF,LIST=ALL
//LAUF      PROC
//GO        EXEC PGM=EINS,REGION=88K
//SYSIN     DD  DSN=SYS1.PROCLIB(KARTEN),DISP=SHR
./ ADD      NAME=KARTEN,LIST=ALL
  ***************
  * DATENKARTEN *
  ***************
/*
```

Das Verfahren heißt LAUF und die Datenkarten sind im Glied KARTEN enthalten. (Falls gewünscht, könnten die Karten auch in eine andere Datei gebracht werden.) Das Programm wird nun mit einer einzigen Karte ausgeführt:

```
//STEP     EXEC LAUF
```

9.3.2 Testen von Verfahren

Man sollte jedes neue Verfahren, nachdem man es zur Bibliothek hinzugefügt hat, austesten. Testfälle dürfen erst eingelesen werden, wenn der Job, der das neue Verfahren hinzufügt, geendet hat. Andernfalls wird nämlich das alte Verfahren getestet. Da viele Jobs von einem bestimmten Verfahren abhängen können, sollte das Verfahren erst ersetzt werden, wenn die Ersetzung ausgetestet worden ist. Es ist am sichersten, wenn man unterstellt, daß jedes neue Verfahren das erste Mal nicht läuft. Es muß vollständig ausgetestet sein, bevor es soweit ist, ein bestehendes Verfahren zu ersetzen.

Zum Testen eines Verfahrens sollte man einen anderen Namen benutzen und das Verfahren unter diesem Namen in die Verfahrensbibliothek einspielen. Läuft es aber, dann kann man es unter seinem richtigen Namen mit ADD oder REPL einspielen. Das Testverfahren wird wieder gelöscht. Man verwendet dazu das in Kapitel 14 beschriebene Dienstprogramm IEHPROGM.

9.3.3 Symbolische Parameter

Wir haben gesehen, wie man katalogisierte Verfahren bei der Ausführung durch das Überschreiben der Parameter in den DD-Karten verändern kann. Das ist aber sehr oft unbequem, weil der Anwender die überschreibbaren Parameter, die Stepnamen im Verfahren, die ddname der zu überschreibenden Karten und ihre Reihenfolge kennen muß. Die symbolischen Parameter bilden eine Alternative für das Verändern der Verfahren zur Ausführungszeit. Das folgende Beispiel veranschaulicht symbolische Parameter.

```
//LAUF     PROC PROGRAM=EINS,UNIT=2311
//GO       EXEC PGM=&PROGRAM
//A        DD   UNIT=&UNIT,SPACE=(TRK,20)
```

Die Zeichen & vor den Variablen PROGRAM und UNIT weisen diese als symbolische Parameter aus. Mit der PROC-Anweisung ist es möglich, den symbolischen Parametern Standardwerte zu unterstellen. Das Verfahren LAUF wird durch eine einzige Karte nämlich // EXEC LAUF aufgerufen. Das Verfahren wird wie folgt interpretiert.

```
//GO        EXEC PGM=EINS
//A         DD   UNIT=2311,SPACE=(TRK,20)
```

Den symbolischen Parametern können aber auch mit der EXEC-Karte andere Werte zugewiesen werden. Ein in der EXEC-Karte kodierter Wert überschreibt für den betreffenden Lauf die Werte der PROC-Karte.

```
//AUFRUF    EXEC LAUF,UNIT=2314,PROGRAM=ZWEI
```

Nun wird das Verfahren wie folgt interpretiert.

```
//GO        EXEC PGM=ZWEI
//A         DD   UNIT=2314,SPACE=(TRK,20)
```

Den symbolischen Parametern müssen stets Werte zugeordnet werden, entweder in der PROC-Karte des Verfahrens oder in der EXEC-Karte, die das Verfahren aufruft. Geschieht das nicht, so erkennt das System einen JCL-Fehler. Die einzige Ausnahme hierzu bildet der DSN-Parameter. Wird DSN=&name kodiert und wird für name weder in der PROC- noch in der EXEC-Karte ein Wert zugeordnet, so interpretiert das System &name als temporäre Datei.

9.3.3.1 *PROC-Anweisung*

Die PROC-Anweisung wird als // *name* PROC *schlüsselwortparameter* kodiert. Der Name ist wahlfrei, er dient dem Programmierer dazu, das Verfahren zu identifizieren. Die Schlüsselwortparameter werden durch Kommas getrennt und können in jeder beliebigen Reihenfolge kodiert werden. Sie ordnen den symbolischen Parametern ihre Standardwerte zu. Die PROC-Anweisung wird nur benötigt, wenn solche Standardwerte zugeordnet werden. Wird sie verwendet, so muß sie die erste Karte des katalogisierten Verfahrens sein.

9.3.3.2 *Kodierung symbolischer Parameter*

Ein symbolischer Parameter beginnt mit dem Zeichen & und kann bis zu 7 alphanumerische Zeichen (A-Z, 0-9) enthalten. Das erste von ihnen

muß ein Zeichen A-Z sein. Symbolische Parameter dürfen nur im Operandenfeld von JCL-Karten kodiert werden, nicht dagegen im Namensfeld, noch im Operationsfeld. PGM, DPRTY, PROC, COND, PARM, ACCT, RD, TIME, REGION und ROLL dürfen nicht als symbolische Parameter verwendet werden, weil sie Schlüsselwortparameter der EXEC-Karte sind. Ein symbolischer Parameter kann selbstverständlich auf mehreren JCL-Karten erscheinen. Wird in einer PROC- oder EXEC-Karte einem symbolischen Parameter mehr als ein Wert zugeordnet, so wird nur der erste benutzt.

```
//LAUF      PROC NAME=LIB1,NAME=LIB2

//STEP1     EXEC PGM=EINS

//A         DD  DSN=&NAME,...

//STEP2     EXEC PGM=ZWEI

//B         DD  DSN=&NAME,...
```

Dem symbolischen Parameter &NAME wird bei jedem Auftreten in diesem Verfahren der Wert LIB1 zugeordnet. Man kann diesen Standardwert für einen Lauf überschreiben. &NAME wird dann bei jedem Auftreten verändert. Würde man symbolische Parameter nicht verwenden, so müßte man jede DD-Karte einzeln überschreiben.

Der einem symbolischen Parameter zugeordnete Wert kann beliebige Länge haben, darf aber kein Zeichen & enthalten. Bei einem Hochkomma im zuzuordnenden Wert erfolgt keine Ersetzung. Werte, die Sonderzeichen, das Zeichen & ausgenommen, enthalten (Leerstelle, . / ' ( ) * + - =), müssen in Hochkommas eingeschlossen werden. Ein Hochkomma wird durch zwei aufeinander folgende Hochkommas dargestellt.

```
//LAUF      PROC NAME='SYS1.PROCLIB',TAPE='2400-6'
```

Ein Wert darf auf einer Folgekarte nicht fortgesetzt werden. Das folgende Beispiel ist falsch.

```
//LAUF      PROC NAME='SYS1.,
//               PROCLIB',TAPE='2400-6'
```

Sie sollte wie folgt kodiert werden.

```
//LAUF      EXEC NAME='SYS1.PROCLIB',
//               TAPE='2400-6'
```

Symbolische Parameter können mit regulären Parametern oder Teilen von regulären Parametern verknüpft werden. Der durch eine solche

Verknüpfung erzeugte Text darf 120 Zeichen nicht überschreiten. (Die ganze DD-Karte kann natürlich mehr Zeichen enthalten.) Es ist auch möglich, zwei symbolische Parameter zu verknüpfen.

```
//LAUF      PROC VOLS=',3,4,',NAME=A,IN='NEW,',OUT=KEEP
                .
                .
                .
//A         DD   DSN=SYS1.&NAME,DISP=(&IN&OUT),VOL=SER=(2&VOLS)
```

Die DD-Karte wird wie folgt interpretiert.

```
//A         DD   DSN=SYS1.A,DISP=(NEW,KEEP),VOL=SER=(2,3,4)
```

Folgt der Text einem symbolischen Parameter, so muß er durch einen Punkt (.) abgetrennt werden (der Punkt wird nicht als Teil des Textes angesehen). Der Punkt ist wahlfrei, falls der Text mit einem Sonderzeichen außer Punkt oder linker Klammer beginnt (,/')*&+-=).

```
//LAUF      PROC TAPE=2400,NAME=SYS1
                .
                .
                .
//A         DD   DSN=&NAME..P,UNIT=&TAPE-2
```

Die DD-Karte wird wie folgt interpretiert.

```
//A         DD   DSN=SYS1.P,UNIT=2400-2
```

Symbolische Parameter werden gelöscht, indem entweder in der PROC- oder EXEC-Karte das Schlüsselwort mit einem Gleichheitszeichen, ohne die Angabe eines Wertes, kodiert wird.

```
//LAUF      PROC VOLS=,NAME=PROCLIB
                .
                .
                .
//A         DD   DSN=SYS1.&NAME,VOL=SER=(2&VOLS)
```

Die DD-Karte wird wie folgt interpretiert.

```
//A         DD   DSN=SYS1.PROCLIB,VOL=SER=(2)
```

Ein Trennungszeichen, wie ein führendes oder nachfolgendes Komma, das neben einem symbolischen Parameter steht, wird bei der Löschung des Parameters nicht mitentfernt.

```
//LAUF      PROC VOLS=
                .
                .
                .
//A         DD   ...,VOL=SER=(2,&VOLS),...
```

Die folgende Interpretation dieser DD-Karte wird vom System als JCL-Fehler erkannt. Der Step bricht ab.

```
//A        DD  ...,VOL=SER=(2,),...
```

Die symbolischen Parameter können in der PROC- oder EXEC-Karte in beliebiger Reihenfolge kodiert werden. Sind NAME und VOLS symbolische Parameter in einem FORTRAN-Compile-Link-Go-Verfahren, so kann wie folgt kodiert werden.

```
//AUFRUF   EXEC FORTCLG,NAME='SYS1.FORTERR',
//              REGION.FORT=104K,VOLS=500,COND.GO=EVEN
```

Alle in der PROC- oder EXEC-Karte aufgeführten und mit einem Wert versehenen symbolischen Parameter müssen natürlich auch in dem Verfahren vorkommen. Das folgende Verfahren ist fehlerhaft, weil DISK als symbolischer Parameter in dem Verfahren nicht vorkommt.

```
//LAUF     PROC DISK=2311
//GO       EXEC PGM=PROGM1
//A        DD  UNIT=2311,SPACE=(TRK,20)
```

Dieser Fehler kann behoben werden, indem man die DD-Karte im Verfahren anders kodiert.

```
//A        DD  UNIT=&DISK,SPACE=(TRK,20)
```

Dieser Fehler läßt sich aber auch verbessern, indem man den UNIT-Parameter mit einer DD-Karte überschreibt.

```
//AUFRUF   EXEC LAUF
//GO.A     DD  UNIT=&DISK
```

Das Beispiel zeigt, daß symbolische Parameter auf überschreibenden DD-Karten kodiert werden können. Auch DD-Karten, die symbolische Parameter enthalten, können überschrieben werden, wenn man achtgibt und keinen symbolischen Parameter überschreibt, dem ein Wert zugeordnet wurde. Wenn man im oben angeführten Verfahren die Verbesserung UNIT=&DISK durchführt, bringt der folgende Step einen Fehler, weil &DISK in der PROC-Karte einen Wert zugeordnet erhält, jedoch nicht mehr im Verfahren erscheint.

```
//AUFRUF   EXEC LAUF
//GO.A     DD  UNIT=2311
```

9.3.3.3 *Anwendung symbolischer Parameter*

MFT- und MVT-Systeme können mehrere Verfahrensbibliotheken enthalten. Wollen wir das vorher in diesem Kapitel beschriebene Verfahren ADDPROC anwenden, um ein Verfahren zu einer anderen Bibliothek als der SYS1.PROCLIB hinzuzufügen, so müssen wir den DSN-Parameter der SYSUT1 DD-Karte überschreiben. Nehmen wir an, es soll ein Verfahren zur SYS1.ALTPROC hinzugefügt werden. (Kapitel 11 zeigt an einem Beispiel, wie eine solche Bibliothek erzeugt wird.)

```
//AUFRUF    EXEC ADDPROC
//GO.SYSUT1 DD   DSN=SYS1.ALTPROC
//GO.SYSIN  DD   DATA
  ***************
  * DATENKARTEN *
  ***************
/*
```

Das geht gut, solange wir weder den Stepnamen, den ddname noch den zu überschreibenden Parameter vergessen oder versehentlich GO.SYSUT1 hinter GO.SYSIN legen. Eine bessere Methode ist es, für den Bibliotheksnamen einen symbolischen Parameter einzuführen, dem man als Wert SYS1.PROCLIB unterstellt.

```
//AUFRUF    EXEC ADDPROC
//GO.SYSIN  DD   DATA
./ REPL          NAME=ADDPROC,LIST=ALL
//ADDPROC   PROC NAME='SYS1.PROCLIB'
//GO        EXEC PGM=IEBUPDTE,REGION=24K
//SYSPRINT  DD   SYSOUT=A
//SYSUT1    DD   DSN=&NAME,DISP=OLD
//SYSUT2    DD   DSN=*.SYSUT1,DISP=OLD
/*
```

ADDPROC wird durch diesen Step in der Bibliothek ersetzt und kann nun dazu benutzt werden, Verfahren zu jeder beliebigen Bibliothek hinzuzufügen. Keiner, der das alte Verfahren benutzt, braucht seine JCL zu ändern, denn der dem symbolischen Parameter unterstellte Wert ist SYS1.PROCLIB. Um ein Verfahren zu SYS1.ALTPROC hinzuzufügen, muß folgendes kodiert werden.

```
//AUFRUF   EXEC ADDPROC,NAME='SYS1.ALTPROC'
//GO.SYSIN DD  DATA
  ***************
  * DATENKARTEN *
  ***************
/*
```

Ein letztes Beispiel soll die Nützlichkeit symbolischer Parameter verdeutlichen. Der Linkage Editor-Step enthält gewöhnlich eine DD-Karte SYSLIB zur Beschreibung von Bibliotheken, die der Linkage Editor nach Unterprogrammen durchsuchen soll. SYSLIB beschreibt oft mehrere miteinander verknüpfte Unterprogrammbibliotheken.

```
//LKED     EXEC PGM=LINKEDIT
//SYSLIB   DD  DSN=SYS1.FORTLIB,DISP=SHR
//         DD  DSN=SYS1.SSPLIB,DISP=SHR
```

Der Linkage Editor durchsucht beide, die SYS1.FORTLIB und die SYS1.SSPLIB, um die benötigten Unterprogramme zu finden. Nun nehmen wir an, daß dieser LKED-Step zu dem FORTRAN-Compile-Link-Go-Verfahren FORTCLG gehört. Es könnte jetzt sein, daß man seine eigene private Bibliothek auch noch mit SYSLIB verknüpfen möchte.

```
//AUFRUF   EXEC FORTCLG
               .
               .
               .
//LKED.SYSLIB DD
//         DD
//         DD  DSN=PRIVLIB,DISP=SHR
```

PRIVLIB wird auf diese Weise mit SYS1.FORTLIB und SYS1.SSPLIB verknüpft. Man überlege sich aber, was passiert, wenn das FORTCLG Verfahren in der Installation dadurch geändert wird, daß eine Unterprogrammbibliothek hinzugefügt wird. Die veränderte SYSLIB-Karte könnte folgendermaßen aussehen.

```
//SYSLIB   DD  DSN=SYS1.NEWFORT,DISP=SHR
//         DD  DSN=SYS1.FORTLIB,DISP=SHR
//         DD  DSN=SYS1.SSPLIB,DISP=SHR
```

Die PRIVLIB-DD-Karte überschreibt nun die SYS1.SSPLIB DD-Karte. Wenn der Job ein Unterprogramm aus der SYS1.SSPLIB braucht, wird

er fehlerhaft enden, weil ja die Bibliothek SYS1.SSPLIB nicht mehr durchsucht wird. Die aufrufende JCL muß durch das Hinzufügen einer weiteren // DD-Karte geändert werden. Mit symbolischen Parametern kann man dieses Problem sehr viel eleganter lösen. Wir wollen annehmen, daß das Verfahren so kodiert wurde, daß der Benutzer eine private Bibliothek mit einem symbolischen Parameter angeben kann.

```
//FORTCLG   PROC LIB='SYS1.FORTLIB'
                 .
                 .
                 .
//SYSLIB    DD   DSN=SYS1.FORTLIB,DISP=SHR
//          DD   DSN=SYS1.SSPLIB,DISP=SHR
//          DD   DSN=&LIB,DISP=SHR
```

Eine private Bibliothek läßt sich jetzt mit SYSLIB durch folgende Kodierung verknüpfen.

```
//AUFRUF    EXEC FORTCLG,LIB=PRIVLIB
```

Dieses Verfahren läßt sich nicht nur leicht und mit weniger Fehlermöglichkeiten kodieren, sondern, was genau so wichtig ist, man kann in der Installation völlig freizügig weitere Bibliotheken zur SYSLIB hinzufügen, ohne daß irgend jemand seine JCL ändern muß.

9.4 Zusammenfassung

Man kann mehrere Dateien wie eine einzige verarbeiten, indem man sie miteinander verknüpft. Die Dateien müssen gleiche Merkmale aufweisen; es ist jedoch statthaft, sequentielle Dateien mit Gliedern untergliederter Dateien zu verknüpfen.

Alle DD-Karten eines Steps (die STEPLIB DD-Karte eingeschlossen) müssen der EXEC-Karte unmittelbar folgen. Die Reihenfolge der DD-Karten muß unter den folgenden Umständen[1]) eingehalten werden:

1) Wenn in einem Step mehrere DD-Karten den gleichen ddname haben, dann wird nur die erste DD-Karte benutzt und die anderen werden ignoriert.

[1]) Einige dieser Regeln sind noch nicht besprochen worden. Trotzdem werden sie der Vollständigkeit halber hier aufgeführt.

2) Dateien werden in der Reihenfolge ihrer DD-Karten miteinander verknüpft.
3) In MFT- und MVT-Systemen werden SYSOUT-Dateien für die gleiche Klasse in der Reihenfolge ihrer DD-Karten ausgedruckt. In PCP-Systemen wird die Ausgabe dieser Dateien nach Anfall gedruckt.
4) In PCP-Systemen müssen DD * bzw. DD DATA Karten die letzten im Step sein.
5) Katalogisierte Verfahren überschreibende DD-Karten müssen in der gleichen Reihenfolge wie die Karten im Verfahren liegen. Hinzuzufügende DD-Karten müssen den überschreibenden folgen. PCP-Systeme verlangen, daß jede überschreibende oder hinzugefügte DD * bzw. DD DATA Karte die letzte im Step ist.
6) DD-Karten mit dem Rückbezugparameter müssen den DD-Karten folgen, auf die sie sich beziehen.
7) Die JOBLIB DD-Karte muß der JOB-Karte unmittelbar folgen.
8) Die DD-Karte SYSCHK muß hinter der JOB- und JOBLIB-Karte liegen, aber vor der ersten EXEC-Karte.
9) Die DD-Karten für eine index-sequentielle Datei müssen den Indexbereich, den Hauptdaten-Bereich und den Überlauf-Bereich in genau dieser Reihenfolge definieren.
10) DD-Karten, die Plattenspeicherplatz mit dem SPLIT-Parameter anfordern, müssen der DD-Karte folgen, die den Platz zuordnet, der aufgeteilt werden soll.
11) Die PROC-Karte muß, falls sie vorhanden ist, die erste Karte im katalogisierten Verfahren sein.

Die katalogisierten Verfahren vereinfachen die JCL erheblich. Ein katalogisiertes Verfahren kann aus mehreren Jobsteps bestehen. Es darf alle JCL-Karten enthalten. Nicht erlaubt sind lediglich:

- die JOB-Karte,
- die Begrenzungskarte (/*),
- die Nullkarte (//),
- eine katalogisierte Verfahren aufrufende EXEC-Karte,
- JOBLIB DD-Karten,
- DD * und
- DD DATA Karten.

Das IBM-Programm IEBUPDTE fügt Verfahren zu Verfahrensbibliotheken hinzu. In den Anweisungen des katalogisierten Verfahrens ist es möglich, symbolische Parameter zu kodieren, um so die Parameter leicht ändern zu können.

10. DD-Karten für periphere E/A-Geräte

Dieses Kapitel beschreibt die DD-Karten, die für die verschiedensten peripheren E/A-Geräte zum Lesen und Schreiben von Daten erforderlich sind. Es werden die DD-Karten für Dateien im Eingabe- und im Ausgabestrom, für konventionelle und graphische E/A-Geräte erläutert. Ferner werden Pseudodateien sowie der Parameter DDNAME besprochen, mit dem sich die Definition einer Datei hinausschieben läßt.

10.1 Dateien im Eingabestrom: * , DATA

Karten stellen die gebräuchlichste Form der Eingabe dar. Um eine Kartendatei im Eingabestrom zu definieren, kodiert man:

```
//DDNAME   DD  *

  ***************
  * DATENKARTEN *
  ***************

/*
```

Sehr oft wird SYSIN als ddname für eine solche Kartendatei benutzt. Die Karte /* zeigt das Ende des Kartenpaketes an. Bei der Verwendung von DD * dürfen die Datenkarten in den Spalten 1-2 weder // noch /* enthalten. Enthalten die Datenkarten trotzdem in den Spalten 1-2 //, wie es der Fall ist, wenn es sich bei den Datenkarten um JCL handelt, so muß statt * DATA kodiert werden.

```
//DDNAME   DD  DATA

  ***************
  * DATENKARTEN *
  ***************

/*
```

DATA hat die gleiche Bedeutung wie * mit dem Zusatz, daß Datenkarten mit // in den Spalten 1-2 vorkommen dürfen. Beide Parameter * und DATA sind Positionsparameter, sie müssen daher vor jedem anderen Parameter in der DD-Karte kodiert werden. Es ist nicht möglich, DD * und DD DATA Karten in ein katalogisiertes Verfahren einzufügen, denn diese Karten müssen ja unmittelbar vor den Datenkarten im Eingabestrom liegen.

10.1.1 Besondere Regeln für PCP

Programme in einem PCP-System dürfen nur eine DD * oder DD DATA Karte pro Jobstep enthalten. Diese Karte muß außerdem die letzte DD-Karte im Step sein. Die Karten werden direkt vom Leser eingelesen. Damit bestimmt der Kartenleser die Verarbeitungsgeschwindigkeit.

10.1.2 Besondere Regeln für MFT und MVT

Programme im MFT- und MVT-Betrieb können viele DD * und DD DATA Karten enthalten (mit unterschiedlichen ddname natürlich). Das ist möglich, weil die Karteneingabe auf einem Plattenspeicher zwischengespeichert wird, bevor der Job zur Ausführung kommt. Jedes Mal, wenn das System beim Einlesen eine Datenkarte erkennt (das ist jede Karte, die kein // oder /* in den Spalten 1-2 enthält), der keine DD * oder DD DATA Karte vorausgegangen ist, fügt es automatisch eine DD * Karte ein, die den ddname SYSIN enthält. Wird ein katalogisiertes Verfahren aufgerufen, so steht die erzeugte SYSIN Karte dem ersten Step des Verfahrens zur Verfügung. Fehlt die Datei-Ende-Karte /*, so unterstellt das System auch diese, falls die nächste Karte mit // in den Spalten 1-2 beginnt.

```
//STEP1     EXEC PGM=PROGM1

  ***************
  * DATENKARTEN *
  ***************

//STEP2     EXEC PGM=PROGM2
```

Diese beiden Steps sind den folgenden gleichwertig.

```
//STEP1     EXEC PGM=PROGM1
//SYSIN     DD   *
  ***************
  * DATENKARTEN *
  ***************
/*
//STEP2     EXEC PGM=PROGM2
```

Die MFT und MVT Reader blocken normalerweise die eingelesenen Datenkarten um Speicherplatz und E/A-Zeit zu sparen. Manche Programme können keine geblockte Eingabe verarbeiten, daher muß der DCB-Unterparameter BLKSIZE verwendet werden, um das Blocken zu verhindern. Der Parameter BUFNO kann ebenfalls kodiert werden; er gibt die Anzahl der Puffer an. (Wird ein Job über Remote Job Entry eingegeben, so wird BUFNO ignoriert.) Diese beiden DCB-Unterparameter sind die einzigen Parameter, die man auf einer DD * oder DD DATA Karte kodieren darf. Läßt man BLKSIZE und BUFNO weg, so werden die in der Reader-Prozedur enthaltenen Werte unterstellt.

```
//SYSIN     DD   *,DCB=BLKSIZE=80
```

Die Größe eines Blockes wird auf 80 Bytes festgesetzt.

```
//SYSIN     DD   DATA,DCB=(BLKSIZE=800,BUFNO=2)
```

Die Größe eines Blockes beträgt 800 Bytes; das Lesen der Eingabe geschieht mit 2 Puffern.

10.2 Nachträgliche Definition von Dateien: DDNAME

Der Parameter DDNAME=*ddname* wird oft im Zusammenhang mit Dateien im Eingabestrom verwendet. Er erlaubt es, die Definition der Datei bis zu der nachfolgenden DD-Karte mit dem angegebenen ddname zu verschieben. In MFT- und MVT-Systemen können neben dem Parameter DDNAME noch die DCB-Unterparameter BLKSIZE und BUFNO kodiert werden. Andere Parameter oder Unterparameter sind nicht gestattet. Der folgende Step verlangt BLKSIZE und BUFNO in der DD* Karte.

```
//GO        EXEC PGM=PROGM1
//DD2       DD  *,DCB=(BLKSIZE=80,BUFNO=3)
  ***************
  * DATENKARTEN *
  ***************
/*
```

Man kann diesen Step natürlich als Verfahren katalogisieren, darf aber die DD2 DD-Karte nicht mit in das Verfahren einfügen, weil DD * und DD DATA Karten in einem Verfahren nicht gestattet sind. Wenn das Verfahren JOG heißt, würde dieser Step wie folgt lauten:

```
//AUFRUF    EXEC JOG
//GO.DD2    DD  *,DCB=(BLKSIZE=80,BUFNO=3)
  ***************
  * DATENKARTEN *
  ***************
/*
```

Der lange DCB-Parameter macht dieses Verfahren aber sehr unhandlich. Die DD2 DD-Karte läßt sich nun aber doch in das Verfahren einschließen, wenn man den Parameter DDNAME verwendet. Das Verfahren lautet dann:

```
//JOG       PROC
//GO        EXEC PGM=PROGM1
//DD2       DD  DDNAME=DD2,DCB=(BLKSIZE=80,BUFNO=3)
```

Der aufrufende Step ist jetzt einfacher.

```
//AUFRUF    EXEC JOG
//GO.DD2    DD  *
  ***************
  * DATENKARTEN *
  ***************
/*
```

Werden die Unterparameter BLKSIZE und BUFNO auf der durch DDNAME angegebenen DD-Karte kodiert, so überschreiben sie die entsprechenden Unterparameter der Karte DDNAME. Da der DDNAME-Parameter jeden beliebigen Namen angeben kann, kann statt DD2 auch SYSIN kodiert werden.

```
//DD2       DD   DDNAME=SYSIN,DCB=(BLKSIZE=80,BUFNO=3)
```

Der folgende Aufruf führt den Step mit BLKSIZE=800 aus.

```
//AUFRUF    EXEC JOG
//GO.SYSIN DD   *,DCB=BLKSIZE=800
  ***************
  * DATENKARTEN *
  ***************
/*
```

DD-Karten, die einen ddname haben, der mit dem Parameter DDNAME in einem katalogisierten Verfahren übereinstimmt, dürfen vor oder nach überschreibenden oder hinzugefügten DD-Karten liegen. Für den Parameter DDNAME gelten die folgenden Einschränkungen:
In einem Jobstep dürfen maximal 5 Parameter DDNAME enthalten sein. Die angegebenen Werte für die 5 ddname müssen unterschiedlich sein. DDNAME darf in der JOBLIB DD-Karte nicht verwendet werden. Erfolgt von einer späteren Karte aus ein Rückbezug, so muß sich dieser auf die Karte beziehen, die DDNAME enthält, nicht aber auf die Karte, die von DDNAME benannt wird. Die Karte mit dem Rückbezug muß der von DDNAME genannten Karte folgen. Ist das nicht der Fall, so bezieht sich der Rückbezug auf eine Pseudodatei. Wird der Parameter DDNAME in einer verknüpften Datei verwendet, so muß er vor allen anderen nicht benannten DD-Karten liegen sowie vor der in DDNAME genannten DD-Karte. Das Folgende ist gültig.

```
//EINGABE   DD   DSN=SYS1.PROCLIB(LIST),DISP=SHR
//          DD   DDNAME=SYSIN
//          DD   DSN=SYS1.PROCLIB(PRINT),DISP=SHR
```

Die folgenden beiden Dateien sind fehlerhaft beschrieben. Bei EINGABE folgt die Karte mit dem Parameter DDNAME einer nicht benannten DD-Karte. Bei SYSIN liegt die in DDNAME genannte Karte vor der DDNAME Karte.

```
//EINGABE   DD   DSN=SYS1.PROCLIB(LIST),DISP=SHR
//          DD   DSN=SYS1.PROCLIB(PRINT),DISP=SHR
//          DD   DDNAME=SPAETER
//SYSIN     DD   DSN=SYS1.PROCLIB(LIST),DISP=SHR
//          DD   DDNAME=SYSIN
```

Der Parameter DDNAME hat noch eine zusätzliche Bedeutung. Ein PCP-System verlangt nämlich, daß jede DD * oder DD DATA Karte beim Überschreiben von katalogisierten Verfahren als letzte im Eingabestrom erscheint. Daneben gibt es aber die generelle Einschränkung, daß überschreibende DD-Karten in der Reihenfolge kodiert werden müssen, in der sie im Verfahren vorkommen. Man führe sich das folgende Verfahren vor Augen.

```
//VERF      PROC
//GO        EXEC PGM=PROGM1
//DD1       DD  DSN=SYS1.PROCLIB(LIST),DISP=SHR
//DD2       DD  DSN=LIBRARY,DISP=SHR
```

Beide DD-Karten sollen überschrieben werden. Soll DD1 mit Daten aus dem Eingabestrom überschrieben werden, so muß folgendes kodiert werden.

```
//AUFRUF    EXEC VERF
//GO.DD1    DD  DDNAME=SYSIN
//GO.DD2    DD  DISP=OLD
//GO.SYSIN  DD  *
  ***************
  * DATENKARTEN *
  ***************
/*
```

Durch diese Kodierung werden beide Regeln befolgt. Die DD * Karte ist die letzte, wie es PCP verlangt. Die Reihenfolge der Überschreibungen ist die der Karten im Verfahren.

10.3 Dateien im Ausgabestrom: SYSOUT

Der Parameter SYSOUT ist ein bequemes Mittel, um Dateien für einen Drucker, einen Stanzer oder ein anderes Gerät auszugeben. Eine Datei im Ausgabestrom wird auf ein bestimmtes Ausgabegerät geleitet, wenn man folgendes kodiert:

//ddname DD SYSOUT= *klasse*

Als Klasse kann irgendeines der Alphabetzeichen A-Z angegeben werden.[1]) Eine Installation kann ihre eigenen besonderen Ausgabeklassen festsetzen, etwa für besondere Formulare, für umfangreiche Ausgabe, für Druck- und Stanzausgabe, für Ausgabe von großer Priorität und für von den anderen Ausgaben getrennte Ausgaben. Herkömmlicherweise ist SYSOUT=A der Drucker und SYSOUT=B der Stanzer.

```
//SYSPRINT DD   SYSOUT=A
//SYSPUNCH DD   SYSOUT=B
```

Der Operator kann eine Ausgabeklasse einem bestimmten Ausgabegerät zuordnen. Er kann die Ausgabeklassen aber auch auf Magnetband ausgeben, um sie später zu verarbeiten, falls nämlich ein Drucker oder Stanzer ausfällt oder einfach, weil die Ausgabe sehr umfangreich ist. Die Ausgabe vieler Jobs läßt sich auf eine einzige Magnetbandrolle ausgeben, denn das Band wird erst zurückgespult, wenn sein Ende erreicht ist, oder wenn der Operator es entlädt.

10.3.1 DCB-Unterparameter

Im allgemeinen ist der DCB-Unterparameter nicht erforderlich; falls nötig kann er aber trotzdem kodiert werden. In PCP-Systemen muß der DCB-Parameter für das besondere E/A-Gerät gelten. In MFT- und MVT-Systemen dagegen gilt er für den Speicher mit direktem Zugriff, auf dem die Ausgabe zwischengespeichert wird.

```
//SYSPUNCH DD   SYSOUT=B,DCB=MODE=C
```

Der Unterparameter MODE erzeugt in einem PCP-System eine Dualkartenausgabe; er tut das nicht in MFT- oder MVT-Systemen. (Dualkartenausgabe ist eine Einrichtung des Stanzers, nicht aber des Speichers mit direktem Zugriff)

```
//SYSPUNCH DD   SYSOUT=B,DCB=BLKSIZE=800
```

Die Angabe BLKSIZE bewirkt in MFT- und MVT-Systemen eine Blockung der auf Plattenspeicher zwischengespeicherten Stanzausgabe. In PCP-Systemen dagegen wird diese Angabe ignoriert, denn auf einem Stanzer können die Karten nicht geblockt werden.

[1]) Falls nötig, dürfen auch die numerischen Ziffern 0-9 verwendet werden. Allerdings beabsichtigt IBM, sie später innerhalb des Systems zu verwenden.

10.3.2 Besondere Regeln für PCP

PCP-Systeme dürfen gleichzeitig bis zu 8 Ausgabeklassen in Verwendung haben, eine von ihnen muß SYSOUT=A heißen. Werden mehrere Dateien konkurrierend in die gleiche Ausgabeklasse ausgegeben, so ist die direkt auf das Ausgabe-Gerät geschriebene Ausgabe dieser Dateien untereinander vermischt. Wird eine Ausgabeklasse auf Magnetband gelegt, so darf für diese Ausgabeklasse zu jedem Zeitpunkt nur eine Datei eröffnet sein.

10.3.3 Besondere Regeln für MFT und MVT

MFT- und MVT-Systeme dürfen gleichzeitig bis zu 36 Ausgabeklassen in Verwendung haben. Die Ausgabe wird auf Plattenspeicher zwischengespeichert. Die Ausgabe-Writer des Systems geben die Ausgabedateien später auf den geeigneten E/A-Geräten aus. Der Operator muß den Ausgabe-Writer für ein bestimmtes Gerät starten. Jeder Writer kann mehrere Ausgabeklassen ausgeben. Bei der Benutzung einer Ausgabeklasse sollte man sich vergewissern, daß für sie ein Ausgabe-Writer gestartet wurde. Ist das nicht der Fall, so bleiben die Ausgabedateien auf Platte zwischengespeichert. Werden mehrere Dateien für die gleiche Ausgabeklasse ausgegeben, so wird jede Datei in der Reihenfolge der DD-Karten getrennt von den anderen ausgedruckt.

10.3.3.1 *Die Parameter UNIT und SPACE*

In MFT- und MVT-Systemen bewirkt der Parameter UNIT die Zwischenspeicherung der Ausgabe auf einer bestimmten Platteneinheit. Läßt man die Angabe UNIT weg, so wird ein im Reader festgehaltener Wert für die Zwischenspeichereinheit unterstellt. Da die Ausgabedateien für MFT- und MVT-Systeme vor dem Drucken oder Stanzen auf Plattenspeicher zwischengespeichert werden, muß selbstverständlich für jede SYSOUT DD-Karte eine Platzzuordnung (SPACE) vorgenommen werden. Ein bestimmter Wert für diesen zuzuordnenden Platz wird im Reader-Verfahren unterstellt. Man muß den Parameter SPACE aber explizit angeben, wenn mehr oder weniger Platz verlangt werden soll als dieser unterstellte Wert angibt. Wird der Platz während der Ausführung des Steps aufgebraucht, so wird der Step sofort beendet. Die Ausgabedateien aber werden voll ausgedruckt und ausgestanzt, soweit sie eben bis zu dem Zeitpunkt vorhanden waren, zu dem der Platz aufgebraucht worden war.

Der Parameter SPACE kann auch dazu benutzt werden, um die Ausgabemenge eines Jobs zu begrenzen. Wenn jemand beispielsweise befürchtet, daß sein Job während der Druckausgabe in eine endlose Schleife geraten könnte, so kann er Rechnerzeit sparen (und natürlich auch Papier), indem er den Platz, der der Druckdatei zugeordnet wird, begrenzt. In Kapitel 11 wird der Parameter SPACE beschrieben.

10.3.3.2 *Besondere Verarbeitungsarten*

MFT- und MVT-Systeme gestatten es dem Anwender, zum Drucken der Ausgabe ein eigenes Programm anzugeben. Sie geben außerdem die Möglichkeit, besondere Ausgabeformulare zu verlangen. Zu diesem Zweck muß die SYSOUT DD-Karte folgendermaßen kodiert werden:

//ddname DD SYSOUT=(*klasse, programm, formnummer*)

Programm ist der Name eines in SYS1.LINKLIB enthaltenen Anwenderprogrammes, das die Ausgabedateien andruckt. Formnummer ist eine 1- bis 4-stellige Zahl, mit der besondere Ausgabeformulare verlangt werden können (besondere Ausgabekarten, besonderes Ausgabepapier etc.). Jede Installation muß ihre eigenen Konventionen für diese Zahlen aufstellen. Das System fordert den Operator vor dem Drucken oder Stanzen der Datei auf, die besonderen Karten bzw. die besonderen Formulare einzulegen. Wenn der Druckvorgang beendet ist, gibt es Anweisung die ursprünglichen Formulare wieder einzulegen.

```
//DRUCK     DD   SYSOUT=(A,DRKPGM)
```

DRKPGM ist ein in der SYS1.LINKLIB enthaltenes Anwenderprogramm. Das System lädt dieses Programm und übergibt ihm die Kontrolle, damit es die Ausgabedateien drucken oder stanzen kann.

```
//DRUCK     DD   SYSOUT=(A,,1001)
```

Der Operator wird vor dem Drucken aufgefordert, das Formular 1001 einzulegen.

```
//STANZ     DD   SYSOUT=(B,STANZPGM,2)
```

Das Anwenderprogramm STANZPGM stanzt die Ausgabedatei. Der Operator wird vor dem Beginn des Stanzens aufgefordert, die Ausgabekarten der Formnummer 2 einzulegen.

10.4 Konventionelle E/A-Geräte

Unter konventionellen E/A-Geräten versteht man Drucker, Kartenleser und Kartenstanzer, Schreibmaschinenkonsolen und Lochstreifenleser. Wie es der englische Name (Unit Record Devices) ausdrückt, wird zu einem Zeitpunkt ein einziger Satz verarbeitet: Eine Karte gelesen oder gestanzt, eine Zeile eingetippt oder angedruckt, etc.

Die meisten konventionellen Geräte werden in MFT- und MVT-Systemen vom System direkt belegt, so daß man bei ihrer Anforderung sorgsam vorgehen muß. Die Kartenleser sind normalerweise den Reader-Programmen, die Drucker und Stanzer den Writer-Programmen zugeordnet. Da das System einem Job nur ein konventionelles Gerät zuordnen kann, das nicht belegt ist, muß der Operator eventuell einen Reader oder Writer stoppen, um das Gerät frei zu machen. Im normalen Betrieb sollten die Programme den Kartenleser, den Kartenstanzer und den Drucker nicht direkt anfordern, sondern die DD-Karten für die Ein- und Ausgabeströme verwenden (DD *, DD DATA, DD SYSOUT).

Aus Sicherheitsgründen können konventionelle Geräte wie Kartenleser, Kartenstanzer oder Drucker verwendet werden, damit wichtige Daten nicht erst auf Platte zwischengespeichert werden müssen. Die direkte Zuordnung kann auch angezeigt sein, wenn Datenmengen verarbeitet werden sollen, die zum Zwischenspeichern zu groß sind. Die DD-Karte für konventionelle Geräte wird folgendermaßen kodiert:

```
//ddname    DD    UNIT=adresse,DCB=(...),UCS=(...)
```

Der Parameter UNIT kann auch mit Typ oder Gruppe kodiert werden, jedoch wird meistens die Adresse benutzt, da man normalerweise ein ganz bestimmtes E/A-Gerät zuordnen möchte.

10.4.1 DCB-Unterparameter

Der DCB-Unterparameter kann für jedes konventionelle Gerät angegeben werden. Die folgenden DCB-Unterparameter können je nach Gerät ebenfalls kodiert werden.

Drucker: PRTSP steuert den Zeilentransport (0 bis 3, PRTSP=1 wird unterstellt). RECFM beschreibt das Satzformat (F, V, U) und das Steuerzeichen für den Druckvorgang (A für ASA und M für Maschinensteuerzeichen). Im Anhang B werden die Steuerzeichen beschrieben.

Kartenleser und Kartenstanzer: MODE spezifiziert Dualkarteneinrichtung (C) oder EBCDI-Code[2]) (E); MODE=E wird unterstellt. Wird MODE =C kodiert, so müssen die Werte für BLKSIZE, LRECL und BUFL mit 160 angegeben werden[3]). RECFM gibt das Satzformat (F, V, U) und das Steuerzeichen (M oder A) an. Der Parameter STACK wählt eines der Ablagefächer aus (1 oder 2), STACK=1 wird unterstellt.

Lochstreifen: RECFM bestimmt das Satzformat (F oder U), CODE beschreibt den verwendeten Lochstreifen-Code wie folgt: (CODE=I wird unterstellt).

I IBM BCD Lochstreifen- und Übertragungs-Code (8 Spuren).
F Friden-Code (8 Spuren).
B Burroughs-Code (7 Spuren).
C National Cash Register-Code (8 Spuren).
A USASCII (8 Spuren).
T Fernschreib-Code (5 Spuren).
N Ohne Umsetzung.

```
//DD1       DD  UNIT=00E,DCB=PRTSP=3
```

Der Drucker mit der Adresse 00E wird zugeordnet, der Zeilenvorschub wird auf 3 festgesetzt.

```
//DD2       DD  UNIT=2540-2,DCB=(MODE=C,STACK=1,BLKSIZE=160,
//              LRECL=160,BUFL=160)
```

Ein Kartenstanzer wird zum Stanzen von Dualkarten angefordert.

10.4.2 Universaltypensatz

Der Parameter UCS gibt für den Drucker 1403 mit der Einrichtung Universaltypensatz den zu benutzenden Typensatz an. UCS gestattet es, eine Druckkette bzw. Druckglieder für einen bestimmten Zeichensatz in den Drucker einzulegen. Wenn das System für die Installation gene-

[2]) Anmerkung des Übersetzers: EBCDIC ist die Abkürzung für Extended Binary-Coded-Decimal Interchange Code, es handelt sich um den erweiterten 8-Bit-IBM-BCD-Code.

[3]) Mit der Dualkarteneinrichtung wird eine Kartenspalte als 12-stellige Dualzahl aufgefaßt. Jede Spalte wird in 2 Bytes des Kernspeichers eingelesen. Die 6 oberen Zeilen der Karte gelangen rechtsbündig (Bit 2-7) in das erste Byte und die 6 unteren Zeilen rechtsbündig in das zweite Byte. Bit 0 und 1 eines jeden Bytes werden auf null gesetzt.

riert wird, wird ein Zeichensatz ausgewählt und festgelegt. Dieser Zeichensatz wird immer dann benutzt, wenn der UCS-Parameter keinen anderen Zeichensatz angibt. Dieser Zeichensatz bleibt auch nach der Beendigung des Jobs für den Drucker wirksam. Erst wenn eine spätere DD-Karte einen anderen Zeichensatz anfordert, verliert der erste seine Wirksamkeit. Die Anforderung eines bestimmten Zeichensatzes geschieht folgendermaßen:

UCS=(*code*,FOLD,VERIFY)

Code ist 1 bis 4-stellig. Er kennzeichnet den Zeichensatz. Der Benutzer kann seine eigenen Zeichensätze zum System hinzufügen und ihnen einen Code zuordnen. Bei der Generierung können Zeichensätze mit den folgenden Codes in das System eingefügt werden.

AN	alphanumerisch	QN	alphanumerisch (PL/1-wissenschaftlich)
HN	alphanumerisch	RN	FORTRAN-COBOL (kommerziell)
PCAN	alphanumerisch	SN	Drucken von Texten
PCHN	alphanumerisch	TN	Drucken von Texten
PN	alphanumerisch (PL/1)	XN	hochleistungs-alphanumerisch
		YN	hochleistungs-alphanumerisch
QNC	alphanumerisch (PL/1-kommerziell)		

FOLD bewirkt das Umklappen des ersten, zweiten und dritten Quadranten des EBCD-Interchange-Codes in den vierten Quadranten; d.h. die hexadezimalen Zeichen 01, 21, 81 und C1 werden alle vier als große A's angedruckt. Die Angabe VERIFY bewirkt eine Anzeige des Zeichensatzes mit gleichzeitiger Aufforderung zur Freigabe durch den Operator.

```
//DD1       DD  UNIT=1403,UCS=QN
```

Der Zeichensatz QN wird angefordert.

```
//DD2       DD  UNIT=1403,UCS=(PN,FOLD)
```

Der Zeichensatz PN wird angefordert. Alle Zeichen werden in den vierten Quadranten umgeklappt.

```
//DD3       DD  UNIT=1403,UCS=(AN,,VERIFY)
```

Der Zeichensatz AN wird angefordert. Der Operator bekommt eine Anzeige mit der Aufforderung zur Freigabe.

10.5 Graphische Geräte

Die DD-Karte für graphische Geräte wird folgendermaßen kodiert:

```
//ddname   DD   UNIT=adresse, DCB=(...)
```

Im Parameter UNIT kann auch der Typ oder eine Gruppe verlangt werden. Normalerweise fordert der Benutzer graphischer Geräte aber ein bestimmtes Gerät an. Für graphische Geräte am Rechner IBM 1130 sollte im ersten Step UNIT=1130 und in allen nachfolgenden Steps DSN=*.rückbezug verwendet werden. Obgleich zwei DCB-Unterparameter, nämlich, GNCP und GDSORG zulässig sind, ist es selten notwendig, sie zu kodieren.

```
//GRAPHIC DD  UNIT=2E0
```

Das graphische Gerät mit der Adresse 2E0 wird angefordert.

10.6 Pseudodateien: DUMMY, NULLFILE

Einer sequentiellen Datei kann der Status einer Pseudodatei zugeordnet werden. Bei diesem Status werden alle E/A-Operationen übergangen; die Zuordnung eines Gerätes, die Platzzuordnung und die Disposition der Datei werden ignoriert. Der Versuch, eine Pseudodatei zu lesen, führt sofort zum Dateiende-Programmausgang. Ein Schreibbefehl wird ignoriert. Den Status einer Pseudodatei kann man nur sequentiellen Dateien zuordnen.

Man verwendet Pseudodateien, um den Programmablauf auszutesten, ohne daß tatsächlich Dateien verarbeitet werden. In gleicher Weise kann man unerwünschte Ausgabedateien unterdrücken (Drucklisten, gestanzte Ausgabe etc.); man verleiht den Ausgabedateien den Status von Pseudodateien.

Der Status einer Pseudodatei wird verliehen, indem man in der DD-Karte als ersten Parameter DUMMY oder indem man DSN=NULLFILE kodiert. Andere Parameter dürfen daneben durchaus kodiert werden, wie man es für eine tatsächliche Datei tun würde, sie werden hier jedoch ignoriert. BLKSIZE und BUFNO könnten für die OPEN-Routine allerdings nötig sein, damit sie Puffer beschaffen kann. Die folgenden Beispiele veranschaulichen die Verwendung von Pseudodateien.

```
//SYSPRINT DD  DUMMY
```

Die von SYSPRINT erzeugte Ausgabe wird unterdrückt.

```
//SYSIN    DD DUMMY
```

Der Versuch, Daten von SYSIN zu lesen, führt sofort zum Datei-Ende-Programmausgang.

```
//AUSGABE  DD  DSN=NULLFILE,DISP=(,CATLG),UNIT=2314,
//             VOL=SER=DA0200,SPACE=(200,100),
//             DCB=(BLKSIZE=800,BUFNO=2)
```

Der verlangte Datenträger muß nicht geladen werden. Es wird kein Platz zugeordnet, und die Datei wird auch nicht katalogisiert. AUSGABE muß allerdings eine sequentielle Datei definieren.

```
//AUSGABE  DD  DUMMY,DSN=DATEI,DISP=(,CATLG),UNIT=2314,
//             VOL=SER=DA0200,SPACE=(200,100),
//             DCB=(BLKSIZE=800,BUFNO=2)
```

Diese Karte ist der vorangegangenen gleichwertig.

Eine Kette miteinander verknüpfter Dateien wird durch eine dazwischenliegende Pseudodatei abgebrochen. Im folgenden Beispiel wird nur die erste Datei gelesen.

```
//DD1      DD  DSN=DATEI1,DISP=SHR
//        DD  DUMMY
//        DD  DSN=DATEI2,DISP=SHR
```

10.7 Zusammenfassung

Die gebräuchlichsten Formen der DD-Karte für die verschiedenen peripheren E/A-Geräte, für das Hinausschieben der Definition einer Datei und für Pseudodateien sind die folgenden:

1) Dateien im Eingabestrom - die Karten dürfen in den Spalten 1 und 2 weder // noch /* enthalten:

```
//ddname   DD *
         Datenkarten
/*
```

2) Dateien im Eingabestrom - die Karten dürfen in den Spalten 1 und 2 // aber kein /* enthalten:

```
//ddname    DD DATA
          Datenkarten
/*
```

3) Das Hinausschieben der Definition einer Datei:

```
//ddname1   DD DDNAME=ddname2
                :
                :
//ddname2   DD ...
```

4) Dateien im Ausgabestrom - SYSOUT=A ist normalerweise Druckausgabe und SYSOUT=B Stanzausgabe:

```
//ddname    DD SYSOUT=klasse
//ddname    DD SYSOUT=(klasse, programm, formnummer)
```

5) Konventionelle E/A-Geräte:

```
//ddname    DD UNIT=adresse,DCB=(..),UCS=(code,FOLD,VERIFY)
```

6) Graphische Geräte:

```
//ddname    DD UNIT=adresse
```

7) Pseudodateien:

```
//ddname    DD DUMMY,...
//ddname    DD=NULLFILE,...
```

11. Speicher mit direktem Zugriff

Die Speicherung auf einem Speicher mit direktem Zugriff geschieht in Spuren. Die Spur ist sozusagen die Grundeinheit für diese Speicherung; man versteht unter ihr einen Kreis auf der beschreibbaren Plattenoberfläche. Alle Spuren, die senkrecht übereinander liegen und eine zur Rotationsachse parallele Ebene berühren, bilden einen Zylinder. Liegen Daten auf verschiedenen Zylindern, so muß, um sie zu erreichen, der Zugriffsmechanismus bewegt werden. Das bedeutet auf der anderen Seite, daß Daten viel schneller gelesen oder geschrieben werden können, wenn sie auf dem gleichen Zylinder liegen.

Trommeln, die nur einen einzigen Zylinder enthalten, sind die schnellsten E/A-Geräte, können dafür aber die wenigsten Daten aufnehmen. Man benutzt sie für kleine, besonders häufig benutzte Dateien und gewöhnlich für Teile des Systems.

Platten sind wegen ihrer großen Speicherkapazität und ihrer angemessenen Geschwindigkeit die vielseitigsten Speicher. Die IBM-Plattenspeichereinheiten 2311 und 2314 besitzen auswechselbare Plattenstapel. Im allgemeinen wird aber in einer Installation die Verwendung privater oder auszuwechselnder Plattenstapel überwacht, denn der Operator braucht zum Auswechseln eines Plattenstapels im Durchschnitt drei Minuten. Man verwendet Platten zum Speichern von sequentiellen, untergliederten, direkten und index-sequentiellen Dateien. Trotz der Vorzüge einer Platte kann das Band oder der Streifenspeicher für besonders große oder besonders selten benutzte Dateien geeigneter sein.

Der Streifenspeicher ist der langsamste Speicher mit direktem Zugriff, hat aber die größte Speicherkapazität. Streifenspeicher werden zur Speicherung großer und selten benutzter Dateien verwendet. Tabelle 2 gibt eine Übersicht über Geschwindigkeit und Kapazität der verschiedenen Speicher mit direktem Zugriff.

Auf jedem Speicher mit direktem Zugriff wird ein Datenträger-Inhaltsverzeichnis, die sogenannte VTOC (Volume Table of Contents) geführt. Sie verzeichnet die bestehenden Dateien und weist den freien Platz auf dem Datenträger aus. Die Installation muß jeden Datenträger vor seinem Gebrauch initialisieren. Dieser Vorgang heißt DASDI (Direct-Access Storage Device Initialization, das bedeutet Initialisierung eines Speichergerätes mit direktem Zugriff). Man führt ihn mit dem IBM-Dienstprogramm IEHDASDR aus. Das Programm IEHDASDR schreibt den Datenträgerkennsatz, legt die VTOC an, initialisiert jede Spur des Datenträgers und stellt Ausweichspuren zur Verfügung, falls eine Spur schadhaft ist. Ein Datenträger muß im allgemeinen nicht mehrfach initialisiert werden.

Tabelle 2 Speicher mit direktem Zugriff

Einheit	Kapazität (In Millionen von Bytes)	Durchschnittl. Suchzeit (Millisekunden)	Durchschnittl. Rotations-Verzögerung (Millisekunden)	Übertragungsgeschwindigkeit (Bytes/Sekunden)
2301 Trommelspeicher	4.09	0	8.6	1 200 000
2302 Plattenspeicher	112.79[1])	165	16.7	156 000
2303 Trommelspeicher	3.91	0	8.6	312 500
2311 Plattenspeicher	7.25[2])	75	12.5	156 000
2314 Magnetplattenspeicher	29.17[2]), [3])	60	12.5	312 000
2321 Streifenspeicher	39.2[4])	100-650	25	55 000

[1]) Kapazität pro Modul. Modell 3 hat ein Modul, Modell 4 zwei Module.
[2]) Kapazität pro Plattenstapel. Die Plattenstapel sind austauschbar.
[3]) Modell A1 ist eine Einheit mit 8 Plattenstapeln, Modell A2 hat 5 Plattenstapel.
[4]) Kapazität pro Zelle. Die Einheit enthält 10 auswechselbare Zellen.

Die VTOC belegt in der Regel 1 oder 2 Zylinder. Sie liegt am Anfang oder in der Mitte des Plattenstapels. Liegt die VTOC in den unteren Zylindern, so ist der zuordenbare und zusammenhängende Platz am größten. Legt man sie in die Mitte des Datenträgers, so wird die Suchzeit verringert, weil der Zugriffsarm von der VTOC bis zu irgendeiner Datei nur noch die halbe Wegstrecke zurücklegen muß. Die erste Spur des Datenträgers enthält den Datenträgerkennsatz, dieser verweist seinerseits auf die VTOC.

11.1 Platzzuordnung

Legt man eine neue Datei auf einem Speicher mit direktem Zugriff an, so muß ihr auf diesem Platz zugeordnet werden. Das geschieht
- durch die Anforderung einer bestimmten Platzmenge,
- durch das Anfordern einer bestimmten Spur des Datenträgers,
- durch die Anforderung von Zylindern und das Aufteilen ihres Platzes zwischen verschiedenen Dateien oder
- durch die Weitervergabe von Platz einer vorher angelegten Datei.

Der Platz wird auf einem Speicher mit direktem Zugriff immer in Spuren zugeordnet. Es ist dabei gleichgültig, auf welche Weise er beschafft wird. Die kleinste zuordenbare Platzmenge ist daher eine Spur.

Wird Platz verlangt, so durchsucht das System die VTOC nach freiem Platz. Es ordnet diesen Platz zu und schreibt die VTOC fort. Das System versucht stets den Platz in zusammenhängenden Spuren (das sind Spuren im gleichen Zylinder) und in benachbarten Zylindern zuzuordnen. Wird kein freier Platz gefunden, so beenden PCP-Systeme den Job. MFT- und MVT-Systeme dagegen geben an den Operator eine Meldung aus, um ihn zu fragen, ob der Job beendet werden oder ob er warten soll. Der Operator wird den Job beenden müssen (Cancel), wenn es ihm nicht möglich ist, einen anderen Plattenstapel zu laden, oder wenn er das Ende eines anderen Jobs nicht abwarten kann, der seinen Platz freigeben wird.

Plattenspeicherplatz, der in nicht zusammenhängenden Bereichen zugeordnet wurde, wird gelesen und geschrieben als bestünde er aus einem großen Bereich. Das System verwaltet nicht zusammenhängenden Platz völlig selbständig. Allerdings lassen sich Dateien schneller lesen und schreiben, wenn sie im gleichen Zylinder liegen, denn bei nicht zusammenhängenden Bereichen ist immer die relativ langsame Suchbewegung des Zugriffsmechanismus zusätzlich nötig. Ständig benutzten Dateien sollte man daher möglichst zusammenhängenden Platz im gleichen Zylinder zuordnen.

11.1.1 Anforderung von Platz: SPACE

Meist wird der Platz angefordert, indem man vom System eine bestimmte Platzmenge verlangt und es ihm überläßt, geeignete Spuren auszuwählen. In einem Multiprogramming-Betrieb, wo es vorkommt, daß mehrere Jobs gleichzeitig Platz anfordern, weiß das System viel besser, wel-

cher Platz verfügbar ist. Platz wird folgendermaßen in Form von Blökken, Spuren oder Zylindern angefordert:

```
        CYL
        TRK                          index
        blocklänge                   directory
SPACE=(---------,(menge, zuschlag,--------))
```

Die Angaben Blocklänge, TRK oder CYL bedeuten, daß der Platz in Blöcken, Spuren oder Zylindern zugeordnet werden soll. Unter Blocklänge wird die Anzahl der Bytes pro Block angegeben. (Die Blocklänge darf 65 535 nicht überschreiten.)

Die Angabe Menge bestimmt, wieviel Platz in der verlangten Einheit (Blöcke, Spuren oder Zylinder) zugeordnet werden soll. Dieser Platz heißt *Primär-Zuordnung (Primary Allocation)* und wird stets auf einem einzigen Datenträger zugeordnet. Beispielsweise fordert SPACE=(80,2000) 2000 Blöcke von je 80 Bytes an und SPACE=(CYL,20) verlangt 20 Zylinder.

Die Angabe Zuschlag ist die Anzahl der zuzuordnenden Einheiten für den Fall, daß die Primär-Zuordnung aufgebraucht ist. Dieser Platz heißt *Sekundär-Zuordnung (Secondary allocation)*. Beide zusammen, Primär- und Sekundär-Zuordnung können bis zu 16 nicht zusammenhängende Bereiche beschaffen. Die Angaben Menge und Zuschlag müssen zusammengenommen groß genug sein, um den größten zu schreibenden Block aufzunehmen. Ist das nicht der Fall, so wird der Platz zwar zugeordnet, aber wieder freigegeben, falls das System den Block der gegebenen Länge nicht unterbringen kann.

Die Angabe Directory hält Platz für das Inhaltsverzeichnis einer untergliederten Datei frei, das die Namen ihrer Glieder aufnimmt. Ähnlich ist es mit der Angabe Index; sie belegt den Platz für die Index-Tabellen einer index-sequentiellen Datei.

11.1.1.1 *Blockweise Anforderung von Platz: blocklänge*

Die blockweise Platzanforderung ist die bequemste, denn der Block ist ja der Block der Benutzerdatei. Die Blocklänge ist außerdem von der Einheit unabhängig, so daß immer der gleiche Platz zugeordnet wird, egal ob es sich um eine Trommel, eine Platte oder den Streifenspeicher handelt.

Das System errechnet die tatsächliche Anzahl der Spuren, die zugeordnet werden müssen, auf Grund der Anzahl der vollständig auf eine Spur

passenden Blöcke. (Tabelle 3 und 4 beschreiben die Kapazitäten der verschiedenen Speicher mit direktem Zugriff.)

Tabelle 3 ist ein Abdruck aus "Supervisor and Data Management Services". Er wurde mit der Erlaubnis der International Business Machines Corporation hergestellt.

Tabelle 3 Kapazitäten der Speicher mit direktem Zugriff

Einheit	Datenträger	Kapazität einer Spur[1])	Spuren pro Zylinder	Anzahl der Zylinder	Gesamtkapazität[1])
2311	Plattenstapel	3 625	10	200	7 250 000
2314	Plattenstapel	7 294	20	200	29 176 000
2302	Plattenstapel	4 984	46	246	56 398 944
2303	Trommel	4 892	10	80[2])	3 913 600
2301	Trommel	20 483	8	25[3])	4 096 600
2321	Zelle	2 000	20[4])	980[4])	39 200 000

[1]) Die Kapazität wird in Bytes angegeben.
[2]) Anzahl der von einem Zugriffsarm bedienten Zylinder. Jedes Modul besitzt 2 Zugriffsarme.
[3]) Eine 2301 Trommel besitzt 25 logische Zylinder.
[4]) Der Datenträger entspricht der Unterzelle einer Zelle des 2321 Streifenspeichers.

Tabelle 4 ist ein Abdruck aus "Supervisor and Data Management Services". Er wurde mit der Erlaubnis der International Business Machines Corporation hergestellt.

Tabelle 4 Kapazitätsformeln der Speicher mit direktem Zugriff

Einheit	Die für jeden Datenblock benötigte Anzahl von Bytes			
	Blöcke mit Schlüssel		Blöcke ohne Schlüssel	
	Bi	Bn	Bi	Bn
2311	81+1.049(KL+DL)	20+KL+DL	61+1.049(DL)	DL
2314	146+1.043(KL+DL)	45+KL+DL	101+1.043(DL)	DL
2302	81+1.049(KL+DL)	20+KL+DL	61+1.049(DL)	DL
2303	146+KL+DL	38+KL+DL	108+DL	DL
2301	186+KL+DL	53+KL+DL	133+DL	DL
2321	100+1.049(KL+DL)	16+KL+DL	84+1.049(DL)	DL

Bi ist irgendein Block, ausgenommen der letzte auf der Spur.
Bn ist der letzte Block auf der Spur.
DL ist die Datenlänge.
KL Ist die Schlüssellänge.

$$\text{Blöcke/Spur} = 1 + \frac{\text{Spurkapazität} - \text{Bn}}{\text{Bi}}$$

(Abzurunden auf die nächste ganze Zahl.)

Beispielsweise ordnet SPACE=(1000,8) 3 Spuren der 2311 zu, weil jede 2311 Spur 3625 Bytes enthält und weil 3 1000-Byte Blöcke in eine Spur passen. Obwohl die Platzanforderung SPACE=(4000,2) den gleichen Gesamtspeicher verlangt, werden doch 4 Spuren der 2311 zugeordnet, weil 2 Spuren nötig sind, um einen Block von 4000 Bytes aufzunehmen. Die Platzanforderung wird immer auf eine ganze Anzahl von Spuren aufgerundet.

Der in der Blocklänge angegebene Wert sollte dem DCB-Unterparameter BLKSIZE gleich sein. Wird eine Sekundär-Zuordnung nötig, so errechnet das System den zuzuordnenden Platz, indem es den Zuschlag mit dem Unterparameter BLKSIZE multipliziert. Diese Multiplikation erfolgt also nicht mit der im SPACE-Parameter angegebenen Blocklänge.

Besitzen die Blöcke Schlüssel, so sollte KEYLEN=schlüssellänge als DCB-Unterparameter kodiert werden, damit das System den Platz für die Schlüssel hinzurechnen kann. Bei Sätzen vom Format variabel oder undefiniert sollte für den Wert Blocklänge ein Durchschnittswert angegeben werden. Das System berücksichtigt den durch die Einrichtung Folgespur (Track Overflow wird in Kapitel 14 beschrieben) eingesparten Platz nicht. Die später erläuterten Unterparameter CONTIG und ROUND werden benutzt, um zusammenhängenden Platz zuzuordnen.

11.1.1.2 *Spuren- oder zylinderweise Anforderung von Platz: TRK, CYL*

Da der Platz immer in Spuren zugeordnet wird, ist der Benutzer mit TRK in der Lage, eine genau bestimmte Platzmenge anzufordern. TRK ist vom Gerät abhängig: SPACE=(TRK,1) bringt beim 2311-Plattenstapel 3625 Bytes und beim 2314-Plattenstapel 7294 Bytes. TRK könnte bei Dateien herangezogen werden, die die Einrichtung Folgespur (Track Overflow) verwenden, da das System die durch diese Einrichtung gemachten Einsparungen nicht ausgleicht, wenn man die Blocklänge statt TRK benutzt.

CYL fordert ganze Zylinder an. Man erreicht dadurch schnelleren Zugriff, weil unnötige Bewegungen des Zugriffsarms vermieden werden. CYL ist ebenfalls vom Gerät abhängig: SPACE=(CYL,1) bringt beim 2311-Plattenstapel 36 250 Bytes und beim 2314-Plattenstapel 145 880 Bytes. SPACE=(CYL,1) ordnet den gesamten Platz einer Trommel zu, weil diese nur aus einem einzigen Zylinder besteht. Bei index-sequentiellen Dateien muß CYL verwendet werden.

Verwendet man TRK oder CYL, so muß der Platz, den man braucht, erst berechnet werden. Dabei muß der Einheitentyp, die Spurkapazität, die Anzahl der Spuren pro Zylinder, die Anzahl der Zylinder pro Datenträger, die Blocklänge, die Schlüssellänge und der dem Gerät eigene Zuschlag in Rechnung gestellt werden.

Die dem Gerät eigenen Zuschläge für jede Spur sind für Hardware-Daten wie z.B. für Adressmarken und für Klüfte zwischen den Blöcken nötig. Die Zuschläge für das Gerät hängen wieder vom Gerätetyp ab und außerdem davon, ob die Blöcke mit oder ohne Schlüssel geschrieben werden sollen. Um den für jeden Block tatsächlich benötigten Platz zu errechnen, müssen die Formeln der Tabelle 4 benutzt werden.

Enthält eine Datei ohne Schlüssel beispielsweise 100 Blöcke von je 800 Bytes, so läßt sich die Anzahl der Blöcke pro Spur mit den Tabellen 3 und 4 errechnen. Die Formel lautet für eine 2311:

$$1 + \frac{3625 - 800}{61 + 1.049(800)} = 4 \qquad \text{(Blöcke pro Spur)}$$

Da 100 Blöcke gespeichert werden sollen, sind 25 Spuren nötig. An dieser Stelle sollte der Leser sehr wohl in Betracht ziehen, den Platz in Blöcken zu verlangen. Der Rechner führt dann die Berechnungen für ihn durch. Für einen 2311-Plattenstapel sind die Angaben SPACE=(800,100) und SPACE=(TRK,25) gleichwertig.

Im Unit-Parameter sollte kein Gruppenname verwendet werden, der mehr als einen Gerätetyp enthält. Definiert SYSDA nämlich 2311 und 2314 Platteneinheiten, so resultiert die Platzanforderung UNIT=SYSDA, SPACE=(TRK,1) in 3625 Bytes, falls eine 2311 ausgewählt wird; sie resultiert dagegen in 7294 Bytes, falls eine 2314 ausgewählt wurde. Man weiß nicht, welches der Fall war.

11.1.1.3 *Sekundär-Zuordnung: zuschlag*

Wird die Angabe Zuschlag kodiert, und ist die Primär-Zuordnung aufgebraucht, so erfolgt eine Sekundär-Zuordnung. Es wird ein weiterer Bereich, ein sogenannter Extent von der Größe Zuschlag mal angeforderte Einheiten zugeordnet. Das System ordnet maximal insgesamt 16 Extents zu (Primär- und Sekundär-Zuordnung zusammengerechnet); werden mehr notwendig, so wird der Step abnormal beendet.

Beispielsweise erzielt SPACE=(800,(10,20) eine Primär-Zuordnung von 10 800 Bytes langen Blöcken. Jeder weitere zugeordnete Extent enthält

dann 20 Blöcke von je 800 Bytes. Der maximal zuordenbare Platz beträgt demzufolge 10 + 20(15)=310 Blöcke von je 800 Bytes.

Da es manchmal schwer ist, den benötigten Speicherplatz abzuschätzen, gestattet es einem die Sekundär-Zuordnung eine vernünftige Annahme zu treffen, und falls diese Annahme zu niedrig war, trotzdem noch mehr Platz zu bekommen. Die Sekundär-Zuordnung macht späteren Platzerwerb möglich, und erlaubt der Datei auf diese Weise zu wachsen, ohne daß der maximale Platz bei der Erzeugung der Datei sofort festgelegt werden muß. Zu diesem Zeitpunkt benötigt ihn die Datei ja noch nicht.

Die Extents hängen in der Regel nicht zusammen, so daß die Zugriffszeit mit zunehmender Extentanzahl natürlich wächst. Sehr häufig benutzte Dateien sollten deshalb keine Extents haben. Die später beschriebenen Unterparameter MXIG und ALX stellen einen Ausweg bezüglich der Extents dar.

Falls zusammenhängender Speicherplatz nicht verfügbar ist, erfolgt die Primär-Zuordnung in bis zu 5 nicht zusammenhängenden Extents. Jeder dieser Extents verringert die maximal gestattete Zahl der Extents um eins. Falls die Primär-Zuordnung nicht aus einem einzigen Extent besteht, sind 2 bis 5 weitere Extents weniger erlaubt. (Die maximale Zahl der Extents beträgt 16.)

Das System reserviert für Extents keinen Platz, bevor sie nicht benötigt werden. Ist daher die Kapazität des Datenträgers erschöpft, so wird, falls ein Extent angefordert wird, der Step abnormal beendet, weil kein Platz mehr verfügbar ist. Dies ist im Multiprogramming-Betrieb ein bekanntes Problem, da durchaus mehrere Jobs Platz auf einem Public Datenträger anfordern können, und es nicht sichergestellt werden kann, daß dieser Platz verfügbar ist. Eine Lösung dieses Problems besteht darin, die Datei über mehrere Datenträger auszudehnen, indem man sie sich mit Hilfe des VOL-Parameters über mehrere Datenträger erstrecken läßt.

```
//DD1       DD  UNIT=2311,VOL=(,,,2),SPACE=(800,(100,50))
```

Wird eine Sekundär-Zuordnung notwendig und ist der Datenträger, der die Primär-Zuordnung enthält, voll, so wird der Platz auf einem zweiten 2311-Datenträger beschafft.

Die Primär-Zuordnung wird stets auf einem einzigen Datenträger durchgeführt. Dateien über mehrere Datenträger müssen den Unterparameter Zuschlag enthalten, damit Platz auf zusätzlichen Datenträgern zugeord-

net werden kann. Index-sequentielle Dateien dürfen keinen Zuschlag erhalten. Bei Checkpoint-Dateien wird ein Zuschlag ignoriert.

Wurde eine Datei ohne den Unterparameter Zuschlag erzeugt, so kann der SPACE-Parameter später trotzdem einen Zuschlag angeben. Wurde die Datei dagegen mit einem Zuschlag erzeugt, dann kann der SPACE-Parameter ihn mit einem anderen Zuschlagswert überschreiben. In beiden Fällen ist dieser Zuschlag nur für die Dauer des Steps wirksam. Er ändert weder den ursprünglichen Zuschlag noch die Höchstgrenze von 16 Extents. Beispielsweise könnte eine Datei wie folgt erzeugt worden sein:

```
//A  .      DD  DSN=DATEI1,DISP=(,CATLG),UNIT=DISK,
//             SPACE=(100,(10,20))
```

Erfolgen später Hinzufügungen, und möchte man den Zuschlag für den einen Lauf vergrößern, so muß wie folgt kodiert werden:

```
//ADD       DD  DSN=DATEI1,DISP=(MOD,KEEP),SPACE=(100,(0,200))
```

Jeder während dieses Steps zugeordnete Extent ist 200 Blöcke von je 100 Bytes groß. Spätere Läufe werden wieder mit einer Extentgröße von 20 Blöcken je 100 Bytes arbeiten.

Beim Überschreiben einer DD-Karte eines katalogisierten Verfahrens, die den Parameter SPACE enthält, muß man sehr vorsichtig sein. Beschreibt die überschreibende Karte eine bestehende Datei, so sollte man im allgemeinen den SPACE-Parameter nicht kodieren. Ist allerdings eine Sekundär-Zuordnung nötig, so gibt der SPACE-Parameter in der überschriebenen Karte den Zuschlag an. Der bei der Erzeugung der Datei angegebene Zuschlag gilt für die Sekundär-Zuordnung nicht. Man sollte in der überschreibenden Karte SPACE= kodieren, um den SPACE-Parameter zu löschen. Auf diese Art und Weise läßt sich das Problem vermeiden.

11.1.1.4 *Platz für das Inhaltsverzeichnis oder die Indextabelle: directory, index*

Bei untergliederten Dateien muß Platz für ein Inhaltsverzeichnis zugeordnet werden, das die Namen, die Zweitnamen (Alias) und die Lage jedes Gliedes enthält. Der Wert Directory gibt die Anzahl der 256 Byte langen Sätze des Inhaltsverzeichnisses an. Jeder Block enthält im Durchschnitt 5 Eintragungen: Namen bzw. Zweitnamen. (Jedes Glied kann bis zu 16 Zweitnamen (Alias) besitzen.) Um die Anzahl der Direc-

tory-Blöcke abzuschätzen, summiere man alle Namen und Zweitnamen, dividiere sie durch 5 und runde das Ergebnis auf die nächste ganze Zahl auf. Hat beispielsweise eine untergliederte Datei 30 Glieder und haben 4 von ihnen einen Zweitnamen, so ordne man 34/5 oder 7 Directory-Blöcke zu: SPACE=(800,(20,100,7)).

Der Wert für Directory sollte großzügig angenommen werden, damit neue Glieder und Zweitnamen hinzugefügt werden können, denn das Inhaltsverzeichnis kann später nicht mehr erweitert werden. Ist das Inhaltsverzeichnis voll, so muß die Datei reorganisiert werden. (Kapitel 13 beschreibt, wie man eine untergliederte Datei reorganisiert.) Der Platz für das Inhaltsverzeichnis wird ebenfalls in ganzen Spuren zugeordnet, es sei denn, auf der letzten Spur ist noch soviel Platz, daß auf ihr ein Block des ersten Gliedes gespeichert werden kann. Ist das nicht der Fall, so geht dieser Platz verloren. Der Platz für das Inhaltsverzeichnis geht von der Primär-Zuordnung weg. Man muß dies bei der Abschätzung des gesamten Platzes in Rechnung stellen.

Index-sequentiellen Dateien kann man mit dem SPACE-Parameter Platz für die Indextabellen zuordnen. Kapitel 14 beschreibt index-sequentielle Dateien ausführlich.

11.1.1.5 *Besondere SPACE-Optionen: RLSE, CONTIG, MXIG, ALX, ROUND*

Im SPACE-Parameter können mehrere Positionsunterparameter kodiert werden. Sie dienen dazu, den unbenutzten Platz freizugeben und den Platz so anzulegen, daß der Zugriff beschleunigt wird. (Für index-sequentielle Dateien ist nur CONTIG gestattet.)

```
        CYL                                  ALX
        TRK                  index           MXIG
        blocklänge           directory       CONTIG
SPACE=(---------,(menge, zuschlag,--------),RLSE,--------,ROUND)
```

11.1.1.6 *Platzfreigabe: RLSE*

RLSE gibt beim Schließen der Datei allen unbenutzten Platz frei. Dieser Unterparameter gestattet es dem Benutzer, mehr Platz zuzuordnen, als vielleicht benötigt wird, ohne daß deshalb Platz verschwendet wird. Wird beispielsweise SPACE=(800,20,RLSE) kodiert und werden nur 4 Blöcke geschrieben, so wird Platz für die restlichen 16 Blöcke freigegeben. Wird ein Extent zugeordnet und nur teilweise benutzt, so wird

der Rest des Extents ebenfalls freigegeben. Wurde der Platz in Zylindern angefordert, so werden nur unbenutzte Zylinder wieder freigegeben; im anderen Fall werden unbenutzte Spuren freigegeben. (RLSE wird ignoriert, falls TYPE=T im CLOSE-Makro der Assembler-Sprache kodiert wurde.)

Der Platz wird nur freigegeben, falls die Datei nicht leer ist, und falls sie nach dem Eröffnen für Ausgabe wieder geschlossen wurde. RLSE kann auch dazu benutzt werden, um den Platz einer bestehenden Datei freizugeben. Man muß SPACE=(0,0,RLSE) in der DD-Karte, die sich auf die Datei bezieht, kodieren, um den unbenutzten Platz beim Schliessen der Datei freizugeben. (Sollte die bestehende Datei eine Sekundär-Zuordnung erfordern, so kodiert man SPACE=(einheit,(,zuschlag),RLSE).)

Auf den ersten Blick erscheint es so, als sollte man RLSE stets anwenden; das ist aber nicht der Fall, denn es gibt einige Nachteile. RLSE ist sehr langsam; es benötigt viel E/A-Zeit um den Platz freizugeben. Fügt man außerdem in einem nachfolgenden Step Daten hinzu, so müssen diese in einem weiteren Extent untergebracht werden; dadurch wird die Verarbeitungszeit natürlich länger.

Überschreibt man die DD-Karte eines katalogisierten Verfahrens mit einer DD-Karte für eine bestehende Datei, so sollte man sicherstellen, daß SPACE kodiert wird, damit ein in der überschriebenen Karte vielleicht vorhandener SPACE-Parameter gelöscht wird. Tut man das nicht, und enthält der SPACE-Parameter in der überschriebenen DD-Karte den Unterparameter RLSE, so wird beim Schließen der bestehenden Datei der unbenutzte Platz freigegeben.

11.1.1.7 *Minimierung der Zugriffszeit: CONTIG, MXIG, ALX*

CONTIG, MXIG und ALX sind sich gegenseitig ausschließende Positionsparameter, die dazu dienen, zusammenhängenden Platz zuzuordnen. Da zusammenhängender Platz die Bewegung des Zugriffsarms verringert, wird auch die Zugriffszeit herabgesetzt. CONTIG verlangt, daß der Platz nur in zusammenhängenden Spuren und Zylindern zugeordnet wird; das bedeutet, alle Spuren in einem Zylinder sind zusammenhängend, und wird mehr als ein Zylinder verlangt, so sind die Zylinder auch zusammenhängend. Wird die Einrichtung Folgespur (Track Overflow) verwendet, so sollte man stets CONTIG kodieren.

MXIG ordnet den größten zusammenhängenden freien Bereich des Datenträgers zu, der gleich dem verlangten Platz oder größer als der ver-

langte Platz ist. Das kann dazu führen, daß beträchtlich mehr Platz zugeordnet wird, als verlangt wurde (angefangen vom verlangten Platz bis hin zum gesamten Platz des Datenträgers). Deshalb sollte dies möglichst nur für große Dateien als Alternative zur Sekundär-Zuordnung verwendet werden. RLSE kann hier wie sonst auch überflüssigen Platz wieder freigeben.

ALX baut eine Liste der fünf größten zusammenhängenden freien Bereiche auf und ordnet von ihnen alle die Bereiche zu, die dem verlangten Platz gleich oder größer als der verlangte Platz sind. Auf diese Weise kann man einen einzigen Bereich bekommen, der so groß ist wie der ganze Datenträger, aber auch fünf Bereiche, die insgesamt größer als der verlangte Platz sind usw. Da auch ALX unter Umständen viel mehr Platz zuordnet als verlangt wurde, sollte auch dieser Parameter nur für große Dateien und als Alternative zur Sekundär-Zuordnung verwendet werden. Auch hier kann RLSE überflüssigen Platz wieder freigeben.

SPACE=(100,(20,10),,CONTIG)

Der Platz wird nur zugeordnet, falls er zusammenhängend zur Verfügung steht.

SPACE=(100,(1000,,5),RLSE,ALX)

Es werden zwischen 100 000 Bytes und dem ganzen Datenträger in bis zu 5 Bereichen, die alle minimal 100 000 Bytes besitzen, zugeordnet. Überflüssiger Platz wird wieder freigegeben.

SPACE=(100,(1000,10),,MXIG)

Ein zusammenhängender Bereich, in der Größe irgendwo zwischen 100 000 Bytes und dem gesamten Datenträger, wird zugeordnet.

11.1.1.8 *Aufrunden auf ganze Zylinder: ROUND*

Durch das Kodieren von ROUND wird in Blöcken verlangter Platz innerhalb von Zylindergrenzen zugeordnet. Das System errechnet den notwendigen Platz, rundet auf volle Zylinder auf und ordnet ganze Zylinder zu, so daß der zugeordnete Bereich mit der ersten Spur eines Zylinders beginnt und auf der letzten Spur eines Zylinders endet. Sollen die Zylinder zusammenhängend sein, so muß zusätzlich CONTIG kodiert werden. ROUND stellt damit sicher, daß die Daten in einem auf Zylindergrenze liegenden Bereich kommen, der genau die minimal nötige Zylinderanzahl enthält. Das verkürzt die Zugriffszeit, weil die Armbewegungen in diesem Fall ja so gering wie möglich sind.

SPACE=(1000,195,,,ROUND)

Es werden 6 2311-Zylinder oder 2 2314-Zylinder zugeordnet. Es ist durchaus möglich, daß diese Zylinder nicht zusammenhängend sind.

SPACE=(1000,(195,10), RLSE,CONTIG,ROUND)

Es werden 6 2311-Zylinder oder 2 2314-Zylinder zugeordnet. Die Zylinder hängen zusammen, eine Sekundär-Zuordnung ist möglich, und überflüssiger Platz wird freigegeben.

11.1.2 Anforderung bestimmter Spuren: ABSTR

Platz kann zugeordnet werden durch Angabe der Adresse der ersten Spur und durch Angabe der zuzuordnenden Anzahl von Spuren. Auf diese Art bekommt man stets zusammenhängenden Platz. Man sollte diese Form aber nur für Dateien anwenden, die von der Lage unabhängig sind. Manchmal wird das bei sehr stark benutzten Dateien praktiziert. Man legt diese Dateien dann nahe an die VTOC. Der Platz wird nur zugeordnet, wenn alle verlangten Spuren verfügbar sind. Um Spuren mit bestimmter Adresse anzufordern, kodiert man:

index
directory

SPACE=(ABSTR,(*menge, adresse*,--------))

Die Menge gibt die Anzahl der verlangten Spuren an, und die Adresse ist die absolute Adresse der ersten Spur, die verlangt wird. (Die Adresse der ersten Spur eines Datenträgers ist 0, sie kann aber nicht zugeordnet werden, weil sie einen Adresshinweis zur VTOC enthält.) Die Angabe Directory ordnet den Platz für das Inhaltsverzeichnis untergliederter Dateien zu. Der Wert Index fordert Platz für die Index-Tabellen index-sequentieller Dateien an.

SPACE=(ABSTR,(200,2400))

Es werden 200 Spuren angefordert, die erste hat die Adresse 2400.

11.1.3 Zylinderaufteilung zwischen mehreren Dateien: SPLIT

SPLIT verteilt die Spuren eines Zylinders zwischen mehreren Dateien, um so die Verarbeitungszeit von Dateien zu reduzieren, deren Sätze

einander entsprechen. Beispielsweise könnte sich eine Personalstammdatei über mehrere einzelne Dateien erstrecken, eine von ihnen könnte die Namen enthalten, eine andere die Gehälter und eine dritte die Berufserfahrung. Für jeden Namen gibt es einen zugeordneten Satz in der Gehaltsdatei und der Datei Berufserfahrung. Teilt man die Zylinder unter diesen Dateien auf, so können einander zugeordnete Sätze dieser drei Dateien ohne Bewegung des Zugriffsarms gelesen werden.

Um Zylinder aufzuteilen, legt man die zusammengehörenden DD-Karten hintereinander. In der ersten DD-Karte wird der gesamte Platz und der Platz für die erste Datei angegeben. Jede folgende DD-Karte fordert dann ihren Teil vom gesamten Platz an. Um den Platz in Zylindern anzufordern, wird folgendes in der DD-Karte kodiert:

SPLIT=(*anzahl*,CYL,(*menge, zuschlag*))

Die Menge gibt die Gesamtzahl der für alle Dateien zuzuordnenden Zylinder an. Anzahl gibt für jeden Zylinder die Zahl der Spuren an, die der ersten Datei zuzuordnen sind. Der Zuschlag nennt die Zylinderanzahl für die Sekundär-Zuordnung. Die Sekundär-Zuordnung wird nur der Datei gewährt, die ihren Platz aufgebraucht hat. Sie wird nicht mehr zwischen allen Dateien aufgeteilt. Jede nachfolgende DD-Karte enthält die Angabe SPLIT=*anzahl*. Der Wert Anzahl gibt für jeden Zylinder die Zahl der Spuren an, die dieser Datei zugeordnet werden sollen.

```
//DD1       DD  DSN=DAT1,DISP=(,CATLG),UNIT=2314,SPLIT=(5,CYL,1)
```

Ein einziger Zylinder wird allen Dateien zugeordnet. Die Datei DAT1 erhält 5 Spuren dieses Zylinders.

```
//DD2       DD  DSN=DAT2,DISP=(,CATLG),SPLIT=15
```

Der Datei DAT2 werden 15 Spuren des einen Zylinders zugeordnet. Der Parameter UNIT ist nicht mehr nötig.

Soll der Platz in Blöcken verlangt werden, kodiert man wie folgt:

SPLIT=(*prozent, blocklänge, (menge, zuschlag)*)

Die Blocklänge ist der durchschnittliche Wert der Blocklänge, die Menge gibt die Gesamtzahl der Blöcke für alle Dateien an, und Prozent ist der Anteil jedes Zylinders in Prozent, der der ersten Datei zugeordnet werden soll. Der Zuschlag ist die Anzahl der Blöcke für die Sekundär-Zuordnung. Das System rundet bei der Primär- und Sekundär-Zuordnung auf volle ganze Zylinder auf. Jede nachfolgende DD-Karte enthält dann

SPLIT=*prozent;* sie gibt damit für jeden Zylinder den zuzuordnenden Prozentsatz an. (Sollten die Blöcke Schlüssel besitzen, muß natürlich KEYLEN=schlüssellänge kodiert werden.)

```
//DD1       DD  DSN=DAT1,DISP=(,CATLG),UNIT=2314,
//              SPLIT=(20,800,(30,10))
```

Allen Dateien zusammen werden 30 800 Bytes lange Blöcke zugeordnet. Die Sekundär-Zuordnung beträgt 10 800 Bytes lange Blöcke. Die Datei DAT1 erhält auf jedem Zylinder 20 Prozent der Spuren.

```
//DD2       DD  DSN=DAT2,DISP=(,CATLG),SPLIT=80
```

Die Datei DAT2 erhält von jedem Zylinder die verbleibenden 80 Prozent der Spuren.

Die durchschnittliche Blocklänge für mit SPLIT zugeordnete Dateien darf 65 535 Bytes nicht überschreiten. SPLIT darf für index-sequentielle, für untergliederte und für direkte Dateien sowie für Dateien auf Trommeln nicht verwendet werden. Der für SPLIT-Dateien zugeordnete Platz wird erst freigegeben, wenn alle die Zylinder teilenden Dateien gelöscht werden (DELETE).

11.1.4 Weitervergabe von Platz: SUBALLOC

Der Parameter SUBALLOC gestattet es Platz zu reservieren, der anschließend der Reihe nach an mehrere Dateien weiterverteilt werden kann. Beispielsweise könnte eine Installtaion ihren Plattenplatz dadurch überwachen, daß jedem Benutzer eine bestimmte Platzmenge zugeteilt wird. Dieser kann den Platz dann selbst auf seine Dateien weiter aufteilen.

Zuerst muß eine Datei eingerichtet werden, um den Platz für die Weitervergabe zu reservieren. Den Platz kann man durch jede der drei vorher besprochenen Methoden reservieren, nämlich: Durch die Anforderung von Platz, durch die Anforderung bestimmter Spuren bzw. durch die Zylinderaufteilung zwischen mehreren Dateien. Die so angelegte Datei kann nur für die Weitervergabe von Platz verwendet werden, denn der Platz wird, mit ihrer ersten Spur beginnend, weitervergeben. Wenn all ihr Platz aufgebraucht ist, existiert sie nicht mehr. Um den Platz einer Datei weiterzuvergeben, kodiert man wie folgt in der nachfolgenden DD-Karte.

CYL
TRK
blocklänge

SUBALLOC=(---------,(*menge, zuschlag, directory*), *rückbezug*)

Die Angaben Blocklänge, TRK, CYL, Menge, Zuschlag und Directory sind bereits erörtert worden. Der Zuschlag gilt nur für die mit SUBALLOC erzeugte Datei, nicht dagegen für die Ausgangsdatei. Der Rückbezug verweist auf eine vorausgegangene DD-Karte, die die Datei beschreibt, deren Platz weitervergeben werden soll. Das System vergibt Platz nur dann weiter, wenn der weiterzuvergebende Platz zusammenhängend ist. Der durch Weitervergabe zugeordnete Platz kann nicht noch einmal weitervergeben werden. Durch SUBALLOC zugeordneter Platz kann für jede Datei getrennt freigegeben werden, ohne daß all der weitergegebene Platz gleichzeitig auch freigegeben wird.

```
//STEP1    EXEC PGM=PROGM1
//DD1      DD   DSN=PLATZ,DISP=(,KEEP),UNIT=2314,
//             SPACE=(800,100,,CONTIG)
```

Es wird Platz für die Weitervergabe bereitgestellt.

```
//DD2      DD   DSN=DATEI1,DISP=(,CATLG),SUBALLOC=(800,20,DD1)
```

Zwanzig 800 Byte lange Blöcke werden vom Platz der Datei PLATZ der Datei DATEI1 zugeordnet. Der Parameter UNIT ist nicht erforderlich.

```
//STEP2    EXEC PGM=PROGM2
//DD3      DD   DSN=DATEI2,DISP=(,CATLG),
//             SUBALLOC=(800,(30,20),STEP1.DD1)
```

Dreißig 800 Byte lange Blöcke werden vom Platz der Datei PLATZ der Datei DATEI2 zugeordnet. Eine Sekundär-Zuordnung von 20 800 Byte Blöcken ist bei Bedarf möglich.

11.2 Datei-Kennsätze: LABEL

Der Parameter LABEL muß kodiert werden, falls die Plattendateien neben den Standard-Kennsätzen noch Anwender-Kennsätze (User Label)

besitzen. Man gibt sie an, indem man LABEL=(,SUL) kodiert. Der LABEL-Parameter ist für Band-Kennsätze (Kapitel 12) und für den Schutz von Dateien (Kapitel 14) gebräuchlicher. Da Anwender-Kennsätze selten verwendet werden, wird die Beschreibung des LABEL-Parameters auf diese späteren Kapitel verschoben.

11.3 Dateien auf Speichern mit direktem Zugriff

Um eine Datei auf einem Speicher mit direktem Zugriff zu erzeugen, muß der SPACE-, SPLIT- oder SUBALLOC-Parameter kodiert werden. Es sollten stets auch DSN, UNIT und DISP kodiert werden, es sei denn, sie werden in einigen seltenen Fällen nicht benötigt. Das ist beispielsweise der Fall, wenn VOL=REF kodiert wurde. UNIT ist dann nicht nötig. Der Parameter VOL ist erforderlich, wenn die Datei auf einem bestimmten Datenträger angelegt werden soll, wenn sie auf mehreren beliebigen Datenträgern liegen soll oder wenn ein Private Datenträger geladen werden muß. Die DCB-Unterparameter müssen wie für die Datei erforderlich kodiert werden. RECFM, BLKSIZE, LRECL und BUFNO sind die häufigsten.

Um eine bestimmte Datei wieder aufzufinden, kodiert man DSN, DISP, UNIT und VOL. Die DCB-Unterparameter RECFM, BLKSIZE und LRECL werden aus dem Kennsatz der Datei geholt, müssen also nicht kodiert werden. Eine Ausnahme ist der Parameter BUFNO. Er kommt nicht aus dem Kennsatz und muß daher bei Bedarf kodiert werden. Ist die Datei katalogisiert oder wurde sie übergeben (PASS), so fallen UNIT und VOL weg.

Das folgende Beispiel zeigt die Erzeugung und das Wiederauffinden temporärer und nicht-temporärer Dateien.

```
//STEP1    EXEC PGM=PROGM1
//DD1      DD  DSN=&&TEMP,DISP=(,PASS),UNIT=2311,
//             SPACE=(800,(10,5))
```

Es wird eine temporäre Datei erzeugt und den nachfolgenden Steps übergeben.

```
//DD2      DD  DSN=DAT1,DISP=(,CATLG),UNIT=2314,
//             SPACE=(800,(20,10),RLSE)
```

Es wird eine nicht-temporäre Datei erzeugt und für späteren Gebrauch katalogisiert.

```
//DD3       DD  DSN=DAT2,DISP=(,KEEP),UNIT=2314,
//              VOL=SER=PL0001,SPACE=(TRK,10)
```

Es wird eine nicht-temporäre Datei erzeugt ohne sie zu katalogisieren.

```
//STEP2     EXEC PGM=PROGM2
//DD4       DD  DSN=&&TEMP,DISP=(OLD,DELETE)
```

Die vorhin übergebene Datei wird angesprochen.

```
//DD5       DD  DSN=DAT1,DISP=SHR
```

Die vorher katalogisierte Datei wird angesprochen.

Im nächsten Beispiel soll Platz für eine Datei angelegt werden, die 100 000 Karten mit einem Blockungsfaktor von 10 enthält. Die DCB-Unterparameter sind demnach: RECFM=FB, BLKSIZE=800, LRECL=80. Wir wollen annehmen, daß die Datei sehr oft benutzt wird; der Platz sollte daher zusammenhängend zugeordnet werden, um die Zugriffszeit herabzusetzen.

Zuerst muß eine Speichereinheit ausgewählt werden. Wir nehmen an, daß die Datei auf ständig geladenen 2314 Plattenstapeln mit der Gruppenbeziehung DISK untergebracht werden kann. Weiter soll unterstellt werden, daß auf keinem der Stapel mehr als 30 zusammenhängende Zylinder frei sind. Wieviel Karten kann man in 30 Zylinder unterbringen und wieviel Datenträger werden also nötig sein?

Mit Tabelle 4 läßt sich die Anzahl der Blöcke pro Spur errechnen:

$$1 + \frac{7\,294 - 800}{101 + 1.043(800)} = 7.94$$

Das bedeutet, auf eine Spur passen 7 Blöcke. Da der Zylinder 20 Spuren enthält, nimmt jeder 2314 Zylinder 140 Blöcke auf. 30 Zylinder enthalten dann 4 200 Blöcke oder 42 000 Karten. Das heißt, für die 100 000 Karten sind 3 Datenträger nötig; auf jedem werden 30 Zylinder zugeordnet. Die DD-Karte lautet:

```
//DD1       DD  DSN=KARTEN,DISP=(,CATLG),UNIT=DISK,VOL=(,,,3),
//              SPACE=(800,(4200,4200),RLSE,CONTIG,ROUND),
//              DCB=(RECFM=FB,BLKSIZE=800,LRECL=80)
```

Als letztes Beispiel eine untergliederte Datei. In Kapitel 9 haben wir katalogisierte Verfahren zu einer Verfahrensbibliothek hinzugefügt. Wie würde man den Platz für eine solche Bibliothek zuordnen? Da die Bibliothek Karten enthält, ist LRECL=80. Der Einfachheit halber wollen wir annehmen, daß die Sätze ungeblockt sind.

Als nächstes muß der notwendige Platz abgeschätzt werden. Wir wollen annehmen, daß das durchschnittliche Verfahren 10 Karten enthält und daß wir wahrscheinlich 50 Verfahren haben werden. Für weitere 150 Verfahren soll außerdem Platz vorgesehen werden. Die Primär-Zuordnung muß also für 50 · 10 = 500, und die Sekundär-Zuordnung für 150 · 10 = 1500 80-Byte Blöcke kodiert werden. Da die Sekundär-Zuordnung 15 mal erfolgen kann, sollte jeder Extent der Sekundär-Zuordnung für 1500/15 = 100 80-Byte Blöcke ausgelegt werden. An Platz für das Directory müssen (50 + 150)/5, das sind 40 Blöcke, vorgesehen werden. Folgender Step legt die Bibliothek an:

```
//STEP1    EXEC PGM=IEFBR14
```

Es wird das Pseudoprogramm IEFBR14 benutzt, da ja kein tatsächliches Programm ausgeführt zu werden braucht.

```
//DD1      DD  DSN=PROC,DISP=(,CATLG),UNIT=2311,
//             SPACE=(80,(500,100,40))
```

11.4 Zugeordnete Arbeitsdateien

In MVT-Systemen kann man sogenannte zugeordnete Arbeitsdateien zur Aufnahme temporärer Dateien einrichten. Die Datei wird einem Initiator/Terminator beim Start zugeordnet, sie wird gelöscht, wenn der Initiator/Terminator stoppt. Da ein Initiator/Terminator während der Laufzeit eines Jobs nur diesem einen Job zugeordnet ist, kann dieser Job alle die dem Initiator/Terminator zugeordneten Arbeitsdateien benutzen. Dadurch läßt sich beträchtlich Zeit sparen, denn man braucht den von mehreren Jobs benutzten Dateien - wie z.B. den Arbeitsdateien der Compiler und des Linkage Editor - nur einmal Platz zuordnen. Die zugeordneten Arbeitsdateien mehrerer Initiator/Terminator dürfen im Namen übereinstimmen.

Will man eine zugeordnete Arbeitsdatei benutzen, so muß man ihren Namen, ein oder zwei & vorangestellt, kodieren. Ebenso müssen alle anderen, für neue Dateien notwendige Parameter kodiert werden, z.B. DISP, UNIT, SPACE, DCB etc. Werden sich widersprechende Parameter angegeben, so wird die zugeordnete Arbeitsdatei ignoriert, und es wird statt dessen eine temporäre Datei erzeugt.

SYSUT1 sei eine zugeordnete Arbeitsdatei. Sie wird in dem katalogisierten Verfahren für den Initiator/Terminator wie folgt kodiert:

```
//SYSUT1   DD  UNIT=2314,SPACE=(CYL,2)
```

Mehrere Steps können die gleiche zugeordnete Arbeitsdatei nacheinander benutzen, wenn sie folgende DD-Karte enthalten:

```
//A        DD  DSN=&&SYSUT1,UNIT=2314,SPACE=(CYL,2)
```

In PCP-Systemen lassen sich zugeordnete Arbeitsdateien einrichten, indem man ständige Dateien anlegt. Da zu einer Zeit nur ein einziger Job läuft, kann jeder Job diese ständige Datei wieder benutzen.

11.5 Zusammenfassung

Platz auf einem Speicher mit direktem Zugriff wird durch eine der folgenden Methoden zugeordnet:

1) Anforderung von Platz

```
          CYL
          TRK                          index
          blocklänge                   directory
SPACE=(----------,(menge, zuschlag,--------),

       ALX
       MXIG
       CONTIG
RLSE,------- ,ROUND)
```

2) Anforderung bestimmter Spuren

```
                                  index
                                  directory
SPACE=(ABSTR,(menge, adresse,------- ))
```

3) Zylinderaufteilung

SPLIT=(*anzahl*, CYL, (*menge, zuschlag*))
SPLIT=*anzahl*
oder
SPLIT=(*prozent, blocklänge*, (*menge, zuschlag*))
SPLIT=*prozent*

4) Weitervergabe

CYL
TRK
blocklänge
SUBALLOC=(-------- , (*menge, zuschlag, directory*), *rückbezug*)

12. Magnetbänder

Ein Magnetband ist eine Bandrolle von 0,5 Inch Breite, deren Band auf der einen Seite magnetisierbar beschichtet ist. Eine volle Bandrolle enthält etwa 720 m Magnetband. Das reicht aus, 480 000 Lochkarten, die in allen Spalten Lochungen enthalten, bei einer Blocklänge von 4800 Bytes pro Block, zu speichern. Ein Byte wird quer über die Breite des Magnetbandes gespeichert. Jedes Bit eines Bytes liegt, bezogen auf die Längsrichtung des Bandes, auf einer sogenannten Spur.

Es gibt zwei Methoden, um die Bits auf dem Band zu speichern. Die ältere, immer noch verwendete Methode zeichnet eine binäre Eins als positive Magnetisierung auf und eine binäre Null durch das Fehlen einer solchen. Es wird ein zusätzliches Prüfbit aufgezeichnet, das die Einser-Bits über die Breite des Bandes auf eine gerade oder ungerade Anzahl ergänzt (leider ist keines von beiden Standard). Wenn das Band gelesen wird, prüft die Hardware auf eine gerade oder ungerade Anzahl von Einser-Bits, um festzustellen, ob die Daten einwandfrei sind. Ist das nicht der Fall, wird ein Prüfbitfehler angezeigt. Die neueren System/360-Bandeinheiten zeichnen eine binäre Eins als positive Magnetisierung und eine binäre Null als negative Magnetisierung auf. Diese Methode ist nicht nur zuverlässiger, sondern sie vermeidet auch den Wirrwarr zwischen Prüfung auf gerade und ungerade - hier ist nämlich die Prüfung auf ungerade Standard.

Die IBM-Rechner der zweiten Generation besaßen 6-Bit Bytes, und demzufolge wurden die Daten in 7 Spuren aufgezeichnet. Da die Rechner des System/360 8-Bit Bytes besitzen, zeichnen die neuen Bandeinheiten die Daten in 9 Spuren auf. Man braucht verschiedene Magnetbandeinheiten um 7- oder 9-Spur Bänder zu verarbeiten; die Bandrolle allerdings kann sowohl für 7-Spur als auch für 9-Spur-Aufzeichnungen benutzt werden, solange beides übereinstimmt.

Die Daten werden entlang jeder Spur mit einer Dichte von 200, 556, 800 oder 1600 bits/inch (bpi) geschrieben. Die 7-Spur-Magnetbandeinheiten können 200-, 556- und 800-bpi Bänder verarbeiten. 9-Spur-Bandeinheiten dagegen sind für 800 oder 1600 bpi ausgelegt. (Auf einer 9-Spur-Bandeinheit muß die Einrichtung Zwei-Dichten zusätzlich eingerichtet werden, wenn beide, nämlich 800- und 1600-bpi Bänder verarbeitet werden sollen.) Auf einem Band mit höherer Dichte können mehr Daten gespeichert werden, außerdem ist die Übertragungsgeschwindigkeit höher.

Der benutzbare Teil einer Bandrolle wird durch zwei Reflektormarken angezeigt. Die erste befindet sich etwa 3 m vom Anfang des Bandes entfernt; sie kennzeichnet den Ladepunkt, und hält die ersten 3 Meter zum Einlegen des Bandes frei. Die zweite ist etwa 4,5 m vom Bandende entfernt. Sie signalisiert das Bandende (End of Volume) und erlaubt, einen angefangenen Block vollständig auszuschreiben. Mit dem VOL=SER-Parameter kann man angeben, daß bei Bandende andere Bandrollen geladen werden sollen.

Wird das Band neu beschrieben, so werden die alten, darauf befindlichen Daten automatisch gelöscht. Bevor das Band allerdings beschrieben werden kann, muß der Operator einen schmalen Plastikring, den *Schreibring*, in die dafür vorgesehene Rille der Magnetbandrolle einlegen. Ist der Schreibring nicht eingelegt, so kann das Band zwar gelesen, aber nicht beschrieben werden. Mit dem Schreibring wird das Band gegen unbeabsichtigtes Überschreiben gesichert. Dem Operator muß mitgeteilt werden, ob der Schreibring eingelegt werden darf oder nicht. In der JCL läßt sich das nicht angeben.

Die auf ein Band geschriebenen Blöcke werden durch eine Kluft (Gap) voneinander getrennt, das ist ein Stück unbeschriebenes Band, dessen Länge für 7-Spur-Bänder 0,75 Inch und für 9-Spur-Bänder 0,6 Inch beträgt. Das Ende der Datei bzw. des Bandes wird durch eine Kluft von 3,5 Inch angezeigt. Dieser Kluft folgt ein besonderer, von der Hardware geschriebener Block, die *Bandmarke*. Auf einer einzigen Bandrolle können mehrere Dateien untergebracht werden.

Magnetbänder werden gelesen, indem das Band an dem Lesekopf vorbeigeführt wird. Dabei werden die Daten in den Kernspeicher in aufsteigend-aufeinanderfolgende und zusammenhängende Speicherstellen übertragen. Wird bei der Übertragung ein Fehler festgestellt, so versucht das System, den betreffenden Block wiederholt zu lesen. Erst wenn das nicht gelingt, wird auf einen ständigen E/A-Fehler (Permanent I/O Error) erkannt. Wurde das Ende der Bandrolle oder das Datei-

ende erreicht, so kann das Programm des Anwenders davon verständigt werden. Bänder lassen sich auch rückwärts lesen, indem man das Band in der umgekehrten Richtung laufen läßt. Die Daten werden dabei in aufeinanderfolgende, aber absteigende Speicherstellen übertragen.

Beim Beschreiben des Bandes werden die Daten aus dem Speicher auf das an den Schreibköpfen vorbeigezogene Band übertragen. Die Übertragung erfolgt aus einem zusammenhängenden Bereich in Richtung aufsteigender Kernspeicheradressen. Unmittelbar darauf, wenn das Band den Lesekopf passiert hat, werden die Daten wieder zurückgelesen. Das geschieht, um sicherzustellen, daß sie einwandfrei aufgezeichnet wurden. Man kann Bänder um einzelne Blöcke oder um ganze Dateien vor oder zurücksetzen; man kann sie bis zum Ladepunkt zurückspulen und entladen. Allerdings kann man nur in Assembler-Programmen alle diese Funktionen selbst ausführen.

12.1 Band-Kennsätze: LABEL

Der Parameter LABEL gibt die Art des Kennsatzes an, er nennt die Dateifolgenummer (d.h. die Stellung der Datei auf dem Band in bezug auf andere Dateien) und zeigt an, ob die Datei als Eingabe oder als Ausgabe zu verarbeiten ist. Der vollständige LABEL-Parameter lautet:

```
                                        IN   RETPD=nnnn
                                        OUT  EXPDT=yyddd
LABEL=(dateifolgenummer, typ, PASSWORD,----,-------------)
```

PASSWORD, EXPDT und RETPD sind Dateischutzeinrichtungen; sie werden in Kapitel 14 erläutert. Die allgemeine Form des LABEL-Parameters für Bänder lautet:

```
                                  IN
                                  OUT
LABEL=(dateifolgenummer, typ, ----)
```

12.1.1 Dateifolgenummer

Befindet sich eine Datei nicht als die erste Datei auf dem Band, so muß mit der Dateifolgenummer ihre Stellung auf dem Band angegeben werden, damit das Band richtig vorgesetzt werden kann.

Die Dateifolgenummer ist eine bis 4-stellige Zahl. Sie beschreibt die Stellung der Datei auf dem Band in bezug auf die anderen Dateien des Bandes. Wird die Dateifolgenummer weggelassen oder wird 0 kodiert, so wird vom System 1 unterstellt. (Ist die Dateifolgenummer größer als die Anzahl der Dateien auf dem Datenträger, so wird das Band bis zum Bandende vorgesetzt.) Bei Bändern mit Kennsätzen kann die Dateifolgenummer die relative Stellung der Datei über mehrere Bänder hinweg angeben. Bei Bändern ohne Kennsätze kann sie nur die Stellung der Datei bezüglich jedes einzelnen Bandes angeben.

Ist die Datei katalogisiert, so wird auch die Dateifolgenummer im Katalog festgehalten. Bei Dateien, die übergeben werden (PASS), wird sie allerdings vom System nicht notiert. Aus diesem Grunde muß die Dateifolgenummer in einer DD-Karte explizit angegeben werden, wenn sich diese DD-Karte auf eine übergebene Datei bezieht.

```
//STEP1    EXEC PGM=PROGM1
//DD1      DD  DSN=DAT2,DISP=(OLD,PASS),UNIT=BAND,
//             VOL=SER=BD0421,LABEL=2
//STEP2    EXEC PGM=PROGM2
//DD2      DD  DSN=*.STEP1.DD1,DISP=(OLD,KEEP)
```

Die Karte DD2 bezieht sich auf die erste Datei des Bandes statt auf die in der Karte DD1 beschriebene zweite Datei. DD2 sollte wie folgt kodiert werden:

```
//DD2      DD  DSN=*.STEP1.DD1,DISP=(OLD,KEEP),LABEL=2
```

12.1.2 Art des Kennsatzes: typ

Magnetbänder können einen Datenträgerkennsatz und Datei-Kennsätze enthalten. Das sind entweder die von IBM vorgesehenen Standard-Kennsätze, von der Installation eingeführte Nichtstandard-Kennsätze oder eine Kombination von beiden. Der Datenträgerkennsatz ist ein Block, der in der Installation vor dem Gebrauch des Bandes mit dem IBM-Dienstprogramm IEHINITT auf das Band geschrieben wird. (Die 1- bis 6-stellige Datenträgernummer wird gewöhnlich auch außen auf die Bandrolle aufgeklebt, um dem Operator das Auffinden des Bandes zu erleichtern.) Der Datenträgerkennsatz wird durch keine Bandmarke von der ersten Datei auf dem Band getrennt.

Banddateien mit Kennsätzen bestehen aus dem Vorsatz, einer Bandmarke, der eigentlichen Datei, einer Bandmarke, dem Nachsatz und wieder einer Bandmarke. Hinter der letzten Datei eines Bandes befinden sich statt einer Bandmarke zwei Bandmarken. Der Vorsatz enthält die Beschreibung der Datei durch den Dateinamen, das Satzformat, die Blocklänge und die Satzlänge. Der Nachsatz enthält die gleichen Angaben wie der Vorsatz, so daß das Band auch rückwärts gelesen werden kann.

Wenn die Datei erzeugt wird und Standard-Kennsätze verlangt werden, schreibt das System die Datei-Kennsätze automatisch. Macht man dagegen die Angabe "keine Kennsätze", so werden weder ein Datenträgerkennsatz noch Datei-Kennsätze erzeugt. In diesem Fall wird das Band die erste Datei, eine Bandmarke, die zweite Datei usw. enthalten. Nach der letzten Datei auf dem Band folgen zwei Bandmarken.

Im Unterparameter Typ des LABEL-Parameters können folgende Angaben gemacht werden:

SL	Standard-Kennsätze (wird Typ weggelassen, so werden sie unterstellt)
NSL	Nichtstandard-Kennsätze
SUL	Standard- und Benutzer-(User) Kennsätze
BLP	Kennsatzverarbeitung überspringen
NL	Keine Kennsätze

12.1.2.1 *Standard-Kennsätze: SL*

Kodiert man SL, oder läßt man Typ weg, so unterstellt das System Standard-Kennsätze und liest die Sätze bis zur ersten Bandmarke ein, um zu prüfen, ob es sich um gültige Kennsätze handelt. Anschließend vergleicht es die Datenträgernummer aus dem ersten Kennsatz mit der im Parameter VOL=SER angegebenen. Stimmen die Nummern nicht überein, so wird der Operator angewiesen, das richtige Band zu laden.

Hat das Band keinen Standard-Kennsatz, und wurde mit dem Parameter VOL=SER auch keine bestimmte Bandnummer angefordert, so wird der Operator angewiesen, die Bandnummer für den Kennsatz zur Verfügung zu stellen. Das System beschreibt das Band daraufhin mit den Kennsatzdaten. Hat der eingelesene Kennsatz eine andere als die unterstellte oder in der DD-Karte angegebene Dichte, so wird die Lesedichte umgeschaltet, vorausgesetzt, das Band ist auf einer Bandeinheit mit der Einrichtung "verschiedene Schreibdichten" geladen. Das System versucht da-

raufhin den Kennsatz noch einmal zu lesen. Wurde ein Band geladen, das mit einer anderen als der erwarteten Anzahl von Spuren beschrieben ist, so wird der Job abnormal beendet.

Stimmen die verglichenen Nummern dagegen überein, so setzt das System das Band bis zu der verlangten Datei vor. Falls das Band beschrieben werden soll, wird der im Parameter DSN angegebene Dateiname in den Datei-Kennsatz geschrieben. Soll das Band allerdings gelesen werden, so wird der im DSN-Parameter angegebene Name mit dem Namen aus dem Kennsatz verglichen. Stimmen die Namen nicht überein, so wird das Programm abnormal beendet. Für jede nicht-temporäre Banddatei müssen der Dateiname, die Bandnummer und die Dateifolgenummer zur Verfügung gestellt werden.

Wird beim Beschreiben eines Bandes Bandende festgestellt, so wird auf das Band ein Nachsatz geschrieben, und die Verarbeitung geht mit der nächsten Bandrolle weiter. Standard-Kennsätze werden vor der Verwendung des Bandes mit dem IBM-Dienstprogramm IEHINITT erzeugt und später nicht mehr geändert.

12.1.2.2 *Nichtstandard-Kennsätze: NSL*

Bei NSL wird zur Verarbeitung der Kennsätze die Kontrolle an Routinen der Installation übergeben.

12.1.2.3 *Standard- und Benutzer-Kennsätze: SUL*

SUL ist mit der Angabe SL gleichwertig bis auf die Tatsache, daß bei SUL zusätzlich weitere Kennsätze des Benutzers von bestimmten Routinen der Installation verarbeitet werden.

12.1.2.4 *Kennsatzverarbeitung überspringen: BLP*

Die Angabe BLP setzt das Band auf die richtige Datei vor, ohne daß der Datenträgerkennsatz oder die Datei-Kennsätze geprüft werden. Beim Schreiben der Datei werden keine Kennsätze erzeugt. Beim Lesen der Datei können als natürliche Folge die DCB-Parameter dem Kennsatz nicht entnommen werden. Sie müssen entweder im Programm kodiert werden oder im DCB-Parameter der DD-Karte angegeben werden. BLP muß benutzt werden, wenn man auf gänzlich leere Bänder schreiben will, oder wenn die Bänder Daten enthalten, die mit den neuen im Prüfbit oder in der Spuranzahl nicht übereinstimmen.

Wird ein Band, das Standard-Kennsätze enthält, mit BLP überschrieben, so wird der Datenträgerkennsatz überschrieben. Wird ein Band, das Standard-Kennsätze enthält, mit BLP gelesen, so muß berücksichtigt werden, daß die erste Datei auf dem Band den Datenträgerkennsatz und die Vorsätze der ersten Banddatei enthält. Die zweite Datei auf dem Band enthält die Blöcke der ersten Banddatei, und die dritte Datei auf dem Band wird aus dem Nachsatz der ersten Banddatei gebildet.

Bänder ohne Kennsätze sollten zweckmäßigerweise bei Stepende entladen werden. Man erreicht das durch die Kodierung von DISP=(...,DELETE). Das System verwendet die Bänder dann nicht als Scratch-Bänder. BLP ist eine Option, die bei einem PCP-System generiert werden muß. Bei einem MFT- oder MVT-System muß sie in der katalogisierten Reader-Prozedur angegeben werden. Wurde die Option BLP nicht spezifiziert, kodiert man aber trotzdem BLP, so wird BLP wie NL behandelt.

12.1.2.5 *Keine Kennsätze: NL*

Die Angabe NL ist der Angabe BLP gleichwertig bis auf die Tatsache, daß bei NL der erste Satz des Bandes eingelesen wird, um sicherzustellen, daß dieser Satz kein Datenträgerkennsatz ist. Die Angabe NL kann allerdings unerwartete Resultate zeitigen. Gänzlich leere Bänder, die also auch keine Bandmarken enthalten, werden beim Leseversuch bis zum Bandende vorgesetzt. Enthält das Band alte Daten mit unterschiedlichem Prüfbit oder von unterschiedlicher Spuranzahl, so gibt das einen E/A-Fehler, und das Programm endet abnormal. Um NL zu verwenden, muß man also wissen, was für Daten auf dem Band enthalten sind. Weiß man das nicht, so sollte man BLP benutzen.

Legt man bei der Anforderung von LABEL=(...,SL) aber ein NL-Band ein und wurde mit dem VOL=SER-Parameter kein bestimmter Datenträger angefordert, so wird vom Operator verlangt, die Kennsatz-Angaben zur Verfügung zu stellen. Das System schreibt den Kennsatz dann auf das Band. Enthält das Band aber von vornherein den verlangten Standard-Kennsatz, so wird ein neuer Datei-Kennsatz nur dann geschrieben, wenn der Aufbewahrungszeitraum abgelaufen ist, und wenn das Band keine durch Kennwort geschützte Dateien enthält.

```
//A          DD  UNIT=TAPE,LABEL=(,BLP)
```

Das Band wird ohne Prüfung des Kennsatzes auf die erste Datei des Bandes eingestellt.

```
//B        DD  DSN=DAT1,DISP=OLD,UNIT=2400,VOL=SER=BD0300,
//            LABEL=2
```

Es werden Standard-Kennsätze unterstellt, und das Band wird bis zur zweiten Banddatei vorgesetzt.

```
//C        DD  UNIT=2400,LABEL=(3,NL)
```

Das Band darf keine Kennsätze enthalten. Es wird bis zur dritten Datei vorgesetzt.

```
//D        DD  DSN=DAT2,UNIT=BAND,LABEL=(1,SL)
//E        DD  DSN=DAT2,UNIT=BAND
```

Die DD-Karten D und E sind gleichwertig, denn der in der Karte D kodierte LABEL-Parameter wird unterstellt, falls der Parameter weggelassen wird.

12.1.3 Verarbeitung als Eingabe oder Ausgabe: IN/OUT

Bei der Benutzung der sequentiellen Basis-Zugriffsmethode (BSAM - Basic Sequential Access Method) kann im OPEN-Makro der Assembler-Sprache INOUT kodiert werden. Dadurch wird die Datei als Eingabedatei eröffnet. Sie kann anschließend als Ausgabedatei benutzt werden, ohne daß sie noch einmal eröffnet werden muß. (Für die umgekehrte Anwendung wird OUTIN kodiert.) Durch die Kodierung von LABEL=(,,,IN) oder LABEL=(,,,OUT) in der DD-Karte kann man das INOUT oder OUTIN des OPEN-Makros überschreiben. Damit erreicht man, daß die Datei für Eingabe oder Ausgabe allein eröffnet wird.

Die LABEL-Unterparameter IN und OUT sind vorzugsweise für FORTRAN-Programme bestimmt. FORTRAN eröffnet die Dateien für INOUT bzw. für OUTIN, je nachdem ob das erste Makro für diese Datei READ oder WRITE lautet. Die Dateien werden dadurch sowohl für Eingabe als auch für Ausgabe eröffnet. Das bedeutet, daß der Schreibring stets eingelegt werden muß, auch wenn das Band nur gelesen werden soll. Damit wird die mit dem Schreibring beabsichtigte Wirkung eigentlich hinfällig. In ähnlicher Weise sind beim Lesen von Dateien die Eingriffe des Operators nötig, wenn die Prüfung für den Aufbewahrungszeitraum anspricht (INOUT oder OUTIN vorausgesetzt). Durch das Kodieren von LABEL=(,,,IN) wird die Datei nur als Eingabedatei eröffnet. Das bedeutet, daß der Schreibring nicht eingelegt werden muß, und daß Da-

teien, deren Aufbewahrungszeitraum überschritten wurde, ohne Eingriffe des Operators gelesen werden können.

```
//A        DD   LABEL=(2,SL,,IN),...
```

Diese Datei kann nur gelesen werden.

12.2 DCB-Unterparameter

Die üblichen DCB-Unterparameter wie DSORG, RECFM, BLKSIZE, LRECL und BUFNO können auch für Bänder, sofern notwendig, kodiert werden. Sätze fester, variabler und undefinierter Länge sind möglich. Variable Sätze können auf 7-Spur-Bandeinheiten allerdings nicht geschrieben werden, es sei denn, die Bandeinheiten sind mit der Einrichtung Datenumsetzung ausgerüstet. Die Blöcke sollten wegen des Rauschens wenigstens 18 Byte lang sein. Zur Angabe der Schreibdichte ist der Unterparameter DEN erforderlich; bei 7-Spur-Bändern braucht man den Unterparameter TRTCH.

12.2.1 Bandschreibdichte: DEN

Die folgende Liste zeigt die verschiedenen Werte für DEN.

DEN	7-Spur-Schreibdichten	9-Spur-Schreibdichten
0	200 bpi (Standard)	-
1	556 bpi	-
2	800 bpi	800 bpi (Standard für 800 bpi Laufwerke)
3	-	1600 bpi (Standard für 1600 bpi Laufwerke mit dem Zusatz "verschiedene Schreibdichte")

Für 7-Spur-Bandeinheiten muß der Parameter DEN angegeben werden, es sei denn, die Schreibdichte beträgt 200 bpi. Bei 9-Spur-Bandeinheiten ist DEN nur erforderlich, wenn ein 800 bpi Band auf einer Bandeinheit mit der Einrichtung "verschiedene Schreibdichte" verwendet werden soll.

12.2.2. Bandaufzeichnungs-Technik: TRTCH

Bei 7-Spur-Bandeinheiten wird TRTCH benutzt, um die Werte für die Prüfung, die Datenumsetzung und die Übersetzung festzulegen. Die Prüfung kann auf gerade oder ungerade erfolgen, die Datenumsetzung erlaubt es, Binärdaten auf 7-Spur-Bandeinheiten aufzuzeichnen und die Übersetzung dient zum Lesen oder Schreiben von BCD-Bändern.

Für den Parameter sind folgende Werte möglich:

TRTCH nicht kodiert	Prüfung auf ungerade, Übersetzung aus, Umsetzung aus. (Diese Art der Bandverarbeitung benutzt man, um Daten von Rechenanlagen der 2.Generation zu verarbeiten.)
Schreiben	Nur die 6 rechten Bits eines Bytes werden auf Band geschrieben; die 2 linken Bits werden ignoriert.
Lesen	Jedes Zeichen vom Band wird in die 6 rechten Bits eines Bytes eingelesen. Die 2 linken Bits werden auf null gesetzt.
TRTCH=E	Die Wirkung ist die gleiche wie beim Weglassen von TRTCH, ausgenommen, daß die Prüfung auf gerade erfolgt.
TRTCH=T	Prüfung auf ungerade, Übersetzung ein, Umsetzung aus. (BCD/EBCDIC-Übersetzung)
Schreiben	Jedes 8-Bit EBCDIC-Zeichen wird in ein 6-Bit BCD-Bandzeichen umgewandelt.
Lesen	Jedes 6-Bit Bandzeichen wird in ein 8-Bit EBCDIC-Zeichen umgewandelt.
TRTCH=ET	Die Wirkung ist die gleiche wie bei TRTCH=T, ausgenommen, daß die Prüfung auf gerade erfolgt.
TRTCH=C	Prüfung auf ungerade, Übersetzung aus, Umsetzung ein. (Dieses Verfahren wird benutzt, um Binärdaten auf 7-Spur-Bänder aufzuzeichnen.) Rückwärtslesen erzwingt das Ausschalten der Datenumsetzung.
Schreiben	Drei 8-Bit Bytes (24 Bits) werden als vier 6-Bit Bandzeichen (24 Bits) geschrieben. Sind die zu speichernden Daten kein Vielfaches von 3, so werden je nach Bedarf 2 oder 3 Bandzeichen geschrieben. Die unbenutzten Bits des letzten Zeichens werden auf null gesetzt.

Lesen — Vier 6-Bit Bandzeichen werden als drei 8-Bit Bytes gelesen. Sind die Daten kein Vielfaches von 4, so werden die restlichen 1, 2 oder 3 Zeichen in den dazu notwendigen Speicher gelesen. Die übrig bleibenden Bits des nicht aufgefüllten Bytes werden auf null gesetzt.

12.3 Verwendung von Bändern

Bänder können nur sequentielle Dateien enthalten. Um eine Datei auf Band zu erzeugen, müssen stets die Parameter DSN, DISP, UNIT und VOL kodiert werden. Ausgenommen sind die seltenen Fälle, in denen diese Parameter offensichtlich nicht notwendig sind; so z.B. ist für temporäre Dateien DSN nicht erforderlich. Die DCB-Unterparameter werden je nach Bedarf kodiert. RECFM, BLKSIZE, LRECL, BUFNO, DEN und TRTCH sind die gebräuchlichsten. Der Parameter LABEL muß kodiert werden, sofern die Datei nicht die erste auf einem Band mit Standard-Kennsätzen ist.

Um eine bestehende Banddatei aufzurufen, kodiert man DSN, DISP, UNIT und VOL. Der Parameter LABEL und die DCB-Unterparameter werden nach Bedarf kodiert. Ist das Band ein Band mit Standard-Kennsätzen, so können die DCB-Unterparameter aus dem Kennsatz gelesen werden. (DEN und TRTCH sind allerdings nicht im Datei-Kennsatz verzeichnet.) Ist die Datei katalogisiert oder wurde sie übergeben, so läßt man UNIT und VOL weg. Ist die Datei nicht katalogisiert oder ist sie nicht die erste Datei auf dem Band, so muß mit LABEL ihre Dateifolgenummer angegeben werden. In den folgenden Beispielen wird gezeigt, wie sich Bänder verwenden lassen.

```
//STEP1    EXEC PGM=PROGM1
//A        DD   UNIT=BAND9
```

Die Datei wird als erste auf ein Scratch 9-Spur-Band mit Standard-Kennsätzen geschrieben.

```
//B        DD   DSN=GEHALT,DISP=(,PASS),UNIT=BAND7,VOL=SER=BD0250,
//             LABEL=(2,BLP),DCB=(DEN=2,TRTCH=C)
```

Die Datei wird als zweite Datei auf einem 7-Spur-Band ohne Kennsatz erzeugt und den nachfolgenden Steps übergeben.

```
//C        DD  DSN=DATEI3,DISP=(,CATLG),UNIT=BAND9,
//            VOL=(PRIVATE,RETAIN,SER=BD0451),LABEL=2
```

Die Datei wird als zweite Datei auf einem 9-Spur-Band erzeugt und katalogisiert. Das Band bleibt bei Stepende geladen.

```
//STEP2    EXEC PGM=PROGM3
//D        DD  DSN=*.STEP1.B,DISP=(MOD,KEEP),LABEL=(2,BLP),
//            DCB=(DEN=2,TRTCH=C)
```

Die Datei auf dem 7-Spur-Band wird wieder zugeordnet.

```
//E        DD  DSN=DATEI3,DISP=OLD
```

Die Datei auf dem 9-Spur-Band wird wieder zugeordnet.

12.4 Zusammenfassung

Die bei Magnetbändern gebräuchliche Form des Parameters LABEL lautet:

```
                    SL
                    NL
                    NSL
                    SUL   IN
                    BLP   OUT
LABEL=(dateifolgenummer,----,,----)
```

Die vollständige Form des Parameters LABEL lautet:

```
                    SL
                    NL
                    NSL
                    SUL              IN    RETPD=tage
                    BLP              OUT   EXPDT=yyddd
LABEL=(dateifolgenummer,----,PASSWORD,----, ----------)
```

Die besonderen DCB-Unterparameter für Magnetbänder sind:

DEN=0	200 bpi Bänder, für 7-Spur-Einheiten unterstellt.
DEN=1	556 bpi Bänder
DEN=2	800 bpi Bänder, für 800 bpi 9-Spur-Bandeinheiten unterstellt.

DEN=3	1600 bpi Bänder, für 1600 bpi 9-Spur-Bandeinheiten mit verschiedener Schreibdichte unterstellt.

TRTCH wird nur bei 7-Spur-Bandeinheiten kodiert.

TRTCH nicht kodiert	Prüfung auf ungerade, Übersetzung aus, Umsetzung aus.
TRTCH=E	Die Wirkung ist die gleiche wie beim Weglassen von TRTCH, ausgenommen, daß die Prüfung auf gerade erfolgt.
TRTCH=T	Prüfung auf ungerade, Übersetzung ein, Umsetzung aus. (Beim Schreiben wird EBCDIC in BCD umgewandelt und beim Lesen BCD in EBCDIC.)
TRTCH=ET	Die Wirkung ist die gleiche wie bei TRTCH=T, ausgenommen, daß die Prüfung auf gerade erfolgt.
TRTCH=C	Prüfung auf ungerade, Übersetzung aus, Umsetzung ein. (Zum Schreiben von Binärdaten auf 7-Spur-Bandeinheiten.)

13. Der Linkage Editor

Der Linkage Editor wird, obwohl er nicht zur eigentlichen JCL gehört, für viele JCL-Anwendungen gebraucht. Das Verständnis seiner Funktionen ist Voraussetzung für die Anwendung vieler System/360-Einrichtungen. In diesem Kapitel wird ebenfalls die Verwendung untergliederter Dateien verdeutlicht. Es zeigt wie man Programme als Glieder untergliederter Dateien aufbewahrt und beschreibt, wie man eine Unterprogrammbibliothek erzeugt. Das Kapitel zeigt ferner, wie man eine untergliederte Datei, die ihre Speicherkapazität erschöpft hat, komprimiert oder noch einmal erzeugt.

13.1 Der Linkage Editor

Das Linkage Editor-Programm verarbeitet Objektmodule, Kontrollkarten für den Linkage Editor sowie andere Lademodule, um später ausführbare Lademodule zu erzeugen. Ein einziger Linkage Editor-Step kann mehrere Objekt- und Lademodule verarbeiten, um entweder ein einzelnes oder mehrere Lademodule zu erzeugen.

Ein Objektmodul ist eine sequentielle Datei, die verschiebliche Maschinenbefehle sowie Daten enthält. Es wird von einem Sprachübersetzer erzeugt. Meist werden Objektmodule in eine temporäre Datei gestellt und an einen nachfolgenden Linkage Editor-Step übergeben. Sie lassen sich aber auch für eine spätere Linkage Editor-Verarbeitung in Karten stanzen.

Der Linkage Editor verarbeitet die Objektmodule, um das Lademodul in eine Form zu bringen, die für die Ausführung geeignet ist. Das Lademodul muß als Glied in eine untergliederte Datei gestellt werden, die auf einem Plattenspeicher liegt. Es läßt sich nicht in Karten stanzen

noch vom Band her verwenden. Man stellt das Lademodul in eine temporäre Datei, wenn es nicht wiederholt verwendet werden soll. Man wird es dagegen in eine nicht-temporäre Datei stellen, um das Programm später in den Speicher laden und ohne die Übersetzungs- und Linkage Editor-Steps ausführen zu können. Der Linkage Editor kann bestehende Lademodule mit Routinen, die gewisse Teile des alten Lademoduls ersetzen sollen, zu einem neuen Lademodul verarbeiten.

Die vom Linkage Editor verarbeitete Grundeinheit ist der *Programmabschnitt* (*Control Section*), ein in sich abgeschlossener eine Einheit bildender Programmteil, der aus Instruktionen und Daten besteht. Ein solcher Programmabschnitt ist die kleinste, getrennt verschiebliche Programmeinheit; sie ist in der Regel ein Unterprogramm, ein Verfahren oder ein COMMON-Speicherbereich.

Dem Linkage Editor können mit den entsprechenden DD-Karten mehrere Quellendateien zur Verfügung gestellt werden. Die sogenannte Primäreingabe, mit der DD-Karte SYSLIN beschrieben, ist stets erforderlich. Sie ist sehr oft eine temporäre Datei, die vom Compile-Step übergeben wird. Mit dieser Primäreingabe wird oft eine Eingabedatei aus dem Eingabestrom verknüpft; dadurch lassen sich gleichzeitig neu übersetzte Unterprogramme mit schon vorher gestanzten Objektmodulen in der Eingabe zusammenfassen.

Mit Linkage Editor-Anweisungen lassen sich bei Bedarf mehrere Eingabequellen zusätzlich angeben. Beispielsweise wird das alte Objektmodul als zusätzliche Eingabequelle angegeben, wenn man Unterprogramme, die andere ersetzen sollen, neu übersetzt hat und dem Linkage Editor eingibt. Auf diese Weise können die Unterprogramme, die nicht noch einmal übersetzt werden müssen, in das Lademodul eingeschlossen werden.

Schließlich können Unterprogrammbibliotheken nach benötigten Unterprogrammen durchsucht werden. Beispielsweise können Funktionen, die von FORTRAN-Programmen benutzt werden, automatisch in Bibliotheken aufgesucht werden. Diese Unterprogrammbibliotheken werden mit einer SYSLIB DD-Karte beschrieben. Mehrere von ihnen können miteinander verknüpft werden.

Der Linkage Editor verarbeitet die Primäreingabe zuerst, danach folgen die zusätzlichen Eingaben und zum Schluß werden die Unterprogrammbibliotheken durchsucht, falls noch ungelöste Referenzen vorhanden sind. Werden zwei Module mit dem gleichen Namen angetroffen, so wird nur das erste von ihnen verwendet.

13.1.1 JCL für den Linkage Editor

Das Linkage Editor-Programm heißt LINKEDIT, es ist in der SYS1.LINKLIB enthalten. (Eine Installation kann mehrere Linkage Editor-Programme von verschiedener Größe und mit unterschiedlichen Namen besitzen.)

```
//LKED     EXEC PGM=LINKEDIT,REGION=96K,
                          MAP
                          XREF
//            PARM=(LIST,----,NCAL,LET,HIAR,OVLY)
```

Herkömmlicherweise heißt der Linkage Editor-Step LKED. Der Parameter REGION ist nur in MVT-Systemen erforderlich. Die tatsächlich benötigte Regiongröße kann in den verschiedenen Installationen unterschiedlich sein.

Die PARM-Unterparameter, die in beliebiger Reihenfolge kodierbar sind, rufen verschiedene Optionen auf. LIST druckt die Linkage Editor-Kontrollkarten an und wird normalerweise verwendet. MAP erzeugt eine Speicherübersicht, die die relative Lage der einzelnen Programmabschnitte zeigt. Sie hilft dem Anwender, die für das Programm notwendige Regiongröße abzuschätzen. Die Angabe XREF schließt MAP ein und erzeugt zusätzlich eine Zuordnungstabelle für das Lademodul.

NCAL löscht den automatischen Bibliotheksaufruf. Man benutzt es beim Erzeugen von Unterprogrammbibliotheken, damit jedes Lademodul nur ein einziges Unterprogramm enthält. LET kennzeichnet ein Lademodul, selbst wenn es noch kleine Fehler enthält, als ausführbar. Das ist beispielsweise der Fall, wenn externe Referenzen wegen der Angabe NCAL ungelöst bleiben. In MVT-Systemen bewirkt die Angabe von HIAR, daß Programmabschnitte in den Hierarchie-Speicher gelegt werden. OVLY erlaubt das Anlegen von Überlagerungsstrukturen. (Einige andere Parameter, die über den Rahmen dieses Buches hinausgehen, wurden weggelassen.)

Das Programm LINKEDIT verlangt eine SYSPRINT DD-Karte zur Beschreibung der Protokolliste, eine SYSUT1 DD-Karte zur Anlage einer Scratch Datei und eine SYSLIN DD-Karte für die Primäreingabe. Eine SYSLMOD DD-Karte ist erforderlich, um die Datei zu beschreiben, die das Lademodul enthalten soll. Eine SYSLIB DD-Karte wird für den automatischen Bibliotheksaufruf benötigt. Weitere DD-Karten können zur Beschreibung zusätzlicher Eingabequellen hinzugefügt werden. Das

folgende Beispiel veranschaulicht die Anwendung des Linkage Editor in einem katalogisierten Verfahren für FORTRAN Compile, Link und Go.

```
//FORTCLG   PROC
//FORT      EXEC PGM=IEYFORT,REGION=100K
```

Der FORTRAN G Compiler heißt IEYFORT und ist in der SYS1. LINKLIB enthalten. REGION wird nur in MVT-Systemen benötigt. Der tatsächlich benötigte Kernspeicher kann bei den einzelnen Installationen unterschiedlich sein.

```
//SYSPRINT DD   SYSOUT=A
```

SYSPRINT beschreibt eine Druckdatei.

```
//SYSPUNCH DD   SYSOUT=B
```

SYSPUNCH beschreibt eine Stanzdatei, Sie ist nur nötig, wenn das Objektmodul auch gestanzt werden soll.

```
//SYSLIN    DD  DSN=&&LOADSET,DISP=(MOD,PASS),UNIT=SYSDA,
//              SPACE=(80,(200,100)),DCB=BLKSIZE=80
```

SYSLIN beschreibt die Datei, die das vom Compiler erzeugte Objektmodul enthalten soll. Die Disposition MOD wird verwendet, damit alle Objektmodule bei mehreren Compile Steps in die gleiche sequentielle Datei gelangen.

```
//LKED      EXEC PGM=LINKEDIT,PARM=LIST,COND=(4,LT,FORT),REGION=96K
```

COND bewirkt das Überspringen des Linkage Editor-Steps, ausgenommen der Compile-Step hat einen Rückkehrcode, der kleiner als 4 ist.

```
//SYSLIB    DD  DSN=SYS1.FORTLIB,DISP=SHR
```

SYSLIB beschreibt die Bibliothek für den automatischen Bibliotheksaufruf.

```
//SYSPRINT DD   SYSOUT=A
```

SYSPRINT definiert die Protokolliste.

```
//SYSLIN    DD  DSN=&&LOADSET,DISP=(OLD,DELETE)
//          DD  DDNAME=SYSIN
```

SYSLIN beschreibt die Primäreingabe; die Compiler-Ausgabedatei wird mit einer Datei des Eingabestroms verknüpft.

```
//SYSLMOD  DD  DSN=&&GOSET(GO),DISP=(,PASS),UNIT=SYSDA,
//             SPACE=(1024,(50,20,1)),DCB=BLKSIZE=1024
```

SYSLMOD definiert die Datei, die das erzeugte Lademodul enthalten soll.

```
//SYSUT1   DD  UNIT=(SYSDA,SEP=(SYSLIN,SYSLMOD)),
//             SPACE=(1024,(50,20)),DCB=BLKSIZE=1024
```

SYSUT1 definiert die vom Linkage Editor benutzte Scratch Datei.

```
//GO       EXEC PGM=*.LKED.SYSLMOD,COND=((4,LT,FORT),(4,LT,LKED))
```

Der Step GO führt das vom Linkage Editor erzeugte Programm aus. COND überspringt diesen Step, es sei denn beide, der Compiler- und der Linkage Editor-Step bringen einen Rückkehrcode kleiner als 4.

```
//FT05F001 DD  DDNAME=SYSIN
//FT06F001 DD  SYSOUT=A
//FT07F001 DD  SYSOUT=B
```

Diese Dateien sind für die Ein- und Ausgabedateien des FORTRAN-Programms erforderlich.

Das ganze Verfahren wird nun wie folgt ausgeführt:

```
//AUFRUF   EXEC FORTCLG
//FORT.SYSIN DD  *
  ************************
  * FORTRAN-QUELLENKARTEN *
  ************************
/*
//LKED.SYSIN DD  *
  ******************************
  * KARTEN VON OBJEKTMODULEN *
  ******************************
/*
//GO.SYSIN DD  *
  ***************
  * DATENKARTEN *
  ***************
/*
```

13.1.2 Linkage Editor-Kontrollkarten

Der Linkage Editor gestattet die Eingabe von Kontrollkarten, mit denen besondere Verarbeitungswünsche ausgedrückt werden können. (Einige dieser Karten, die über den Rahmen dieses Buches hinausgehen, werden in der folgenden Diskussion nicht erörtert.) Die Linkage Editor-Kontrollkarten können vor oder nach den Objektmodulen bzw. anderen Kontrollkarten in die Primäreingabe gestellt werden.

1) ENTRY gibt die erste auszuführende Instruktion in einem Lademodul an. Es kann der Name eines Programmabschnitts oder der Name eines Eingangspunktes in einem Programmabschnitt sein. Jedes Lademodul muß selbstverständlich einen Eingangspunkt haben. Fehlt die ENTRY-Karte, so unterstellt der Linkage Editor, daß das erste Byte des ersten Programmabschnitts der Eingangspunkt ist, es sei denn, eine END-Anweisung des Assemblers oder Compilers definiert einen Eingangspunkt. (Der in einer END-Anweisung bezeichnete Eingangspunkt wird nicht festgehalten, falls das Lademodul noch einmal mit dem Linkage Editor verarbeitet wird.) Die ENTRY-Karte wird folgendermaßen kodiert:

   bENTRY name[1])

   Der Name gibt den Namen einer Instruktion und nicht den eines Datenelements an.

   bENTRY SUB1

   Die Ausführung des Programms beginnt mit der Instruktion SUB1.

2) ALIAS gestattet es, sich auf ein Lademodul mit bis zu 16 Zweitnamen zu beziehen. Beispielsweise könnten zwei Unterprogramme in einem einzigen Lademodul einer Unterprogrammbibliothek zusammengefaßt werden, weil die Verwendung des einen Unterprogramms stets auch das andere erforderlich macht. Wird nun dem Lademodul der Name des einen Unterprogramms zugeordnet, und geschieht es, daß das andere Unterprogramm aufgerufen wird, so würde das Lademodul nicht aufgefunden werden. Der Name des zweiten Unterprogramms sollte deshalb ein Zweitname für das Lademodul werden, damit beide aufgefunden werden können, egal welches Unterprogramm gerufen wird.

[1]) Das "b" steht für eine Leerstelle und zeigt an, daß die Karte nicht in Spalte 1 beginnen darf.

Ein Modul, das mit einem Zweitnamen gerufen wird, wird seine Ausführung mit dem externen Namen beginnen, der durch ALIAS angegeben wurde. Ist der Zweitname kein externer Name, so beginnt die Ausführung an dem Haupteingangspunkt des Programms. Die ALIAS-Karte wird wie folgt kodiert:

bALIAS name,...,name[2])

Sechzehn Zweitnamen können auf einer oder mehreren ALIAS-Karten zugeordnet werden.

bALIAS GO,ENTRY1,ENTRY2

Auf das Lademodul kann man sich nun mit den Zweitnamen GO,ENTRY1 und ENTRY2 beziehen.

3) INCLUDE spezifiziert zusätzliche Eingabequellen, die der Linkage Editor mit aufnehmen soll. Die aufzunehmende Datei darf wiederum INCLUDE-Karten enthalten. Allerdings werden die Daten, die dieser INCLUDE-Karte in der angesprochenen Datei folgen, nicht mehr verarbeitet.

Man verwendet INCLUDE-Karten, um das alte Lademodul aufzunehmen, wenn Unterprogramme aus ihm neu übersetzt wurden und nun vom Linkage Editor verarbeitet werden sollen. Der Linkage Editor verarbeitet zuerst die neuen Unterprogramme. Anschliessend nimmt er das alte Lademodul ohne die ersetzten Unterprogramme mit auf. Die INCLUDE-Karte kann auf zwei Arten kodiert werden.

bINCLUDE ddname,...,ddname
oder
bINCLUDE ddname(glied,...,glied),..,ddname(glied,...,glied)

Die Angabe ddname ist der Name der DD-Karte, die die einzuschließende Datei beschreibt. Die einzuschließende Datei kann eine Bibliothek oder eine sequentielle Datei sein, die Objektmodule und Kontrollkarten enthält. Sie kann aber auch eine nur aus Lademodulen bestehende Bibliothek sein. Es sind mehrere INCLUDE-Karten gestattet. Für alle Glieder, die aus Bibliotheken hinzugefügt werden sollen, müssen die Namen kodiert werden. Bei sequentiellen Dateien muß nur der ddname kodiert werden.

---

[2]) Eine Linkage Editor-Karte wird fortgesetzt, indem man die Karte nach einem Komma irgendwo vor Spalte 72 unterbricht, in Spalte 72 irgendein Zeichen ablocht und ab Spalte 16 der nächsten Karte weiterlocht.

bINCLUDE EINGABE1,DD1
bINCLUDE EINGABE2,DD2,(EINS,ZWEI),DD3(SQRT)

Drei in den DD-Karten EINGABE1, DD1 und EINGABE2 beschriebene sequentielle Dateien sollen hinzugefügt werden. Die Glieder EINS und ZWEI aus der in der DD2 Karte beschriebenen Bibliothek, sowie das Glied SQRT aus der in der DD3 Karte beschriebenen Bibliothek werden ebenfalls mit hinzugefügt.

4) LIBRARY benennt Programmabschnitte, nach denen in anderen Bibliotheken als den in der SYSLIB DD-Karte beschriebenen gesucht werden soll. LIBRARY gestattet darüber hinaus externe Referenzen in diesem Step für das Lademodul ungelöst zu lassen.

bLIBRARY ddname(glied,...,glied),..,ddname(glied,...,glied)

Zur Beschreibung der Bibliothek muß eine DD-Karte mit dem angegebenen ddname hinzugefügt werden. Für jeden Programmabschnitt müssen die Namen der Glieder aufgeführt werden, nach denen in der Bibliothek gesucht werden soll.

bLIBRARY (name,...,name)

Die hier namentlich aufgeführten Programmabschnitte bleiben für diesen Lauf ungelöst. Es erfolgt für die benannten Programmabschnitte kein automatischer Bibliotheksaufruf. Beispielsweise könnte in einem Programm auf die Routine einer Bibliothek Bezug genommen werden, ohne daß sie in einem bestimmten Programmlauf angesprochen wird. Da diese Routine also nicht aufgerufen wird, kann die Referenz ungelöst bleiben, um den Speicherplatz, den diese Routine erfordern würde, einzusparen. (In der EXEC-Karte sollte LET kodiert werden.)

bLIBRARY * (name,...,name)

Der hinzugefügte Stern (*) bewirkt, daß die externen Referenzen für die gesamte Lebensdauer des Lademoduls ungelöst bleiben.

Die oben angeführten Parameter können alle auf einer oder mehreren LIBRARY-Karten kodiert werden.

bLIBRARY DD1(SUB1,SUB2),(HALT),*(ALTO)

Nach SUB 1 und SUB2 wird in der Bibliothek gesucht, die in der DD1-Karte beschrieben ist. Für HALT oder ALTO wird

kein automatischer Bibliotheksaufruf vorgenommen. ALTO bleibt darüber hinaus für die gesamte Lebensdauer des Lademoduls ungelöst.

5) HIARCHY bewirkt, daß in MVT-Systemen Programmabschnitte in den Hierarchie-Speicher gelegt werden. Der Parameter REGION muß diesen Speicher anfordern, und HIAR muß in der EXEC-Karte kodiert werden. Die HIARCHY-Karte wird wie folgt kodiert:

bHIARCHY 1,name,...,name

Die benannten Programmabschnitte werden in den Hierarchie-1-Speicher geladen. Mehrere Programmabschnitte können auf einer oder mehreren HIARCHY-Karten angeführt werden. Programmabschnitte, die nicht in der HIARCHY-Karte aufgeführt wurden, werden in den Hauptspeicher geladen.

bHIARCHY 1,SUB1,SUB6

SUB1 und SUB6 werden in den Großraumspeicher geladen.

13.1.3 Erzeugung von Programmbibliotheken

Nehmen wir an, daß ein FORTRAN-Programm übersetzt und in einer Datei namens PROGRAMM aufbewahrt werden soll. Es muß dazu der Name für ein Glied ausgewählt werden, der 1 bis 8 alphanumerische (A-Z, 0-9) oder nationale (@ $ #) Zeichen enthält, von denen das erste alphamerisch (A-Z) oder national sein muß. EIGEN wäre ein passender Name. Das katalogisierte Verfahren FORTCLG kann verwendet werden, allerdings muß die DD-Karte SYSLMOD überschrieben werden, um eine nicht-temporäre Datei anzulegen.

```
//STEP     EXEC FORTCLG
//FORT.SYSIN DD  *
  ************************************
  * KARTEN DES HAUPTPROGRAMMS MAIN *
  * KARTEN DES UNTERPROGRAMMS SUB1 *
  ************************************
/*
//LKED.SYSLMOD DD  DSN=PROGRAMM(EIGEN),DISP=(,CATLG),
//             SPACE=(1024,(10,5,10))
```

Die Datei PROGRAMM wird erzeugt, und das Lademodul wird als Glied mit dem Namen EIGEN in sie hineingestellt. Der Parameter SPACE wird überschrieben, damit eine genauere Platzmenge zugeordnet wird und damit der Platz für das Inhaltsverzeichnis (Directory) größer wird.

```
//LKED.SYSIN DD  *

  ************************************
  * OBJEKTMODUL SUB2 ALS KARTENPAKET *
  ************************************

/*

//GO.SYSIN DD  *

  ***************
  * DATENKARTEN *
  ***************

/*
```

Die drei Routinen MAIN, SUB1 und SUB2 werden zu dem Lademodul EIGEN in der Datei PROGRAMM verbunden. Viele andere Programme können in die Datei PROGRAMM gestellt werden, vorausgesetzt, man gibt ihnen von EIGEN abweichende Namen. Das Programm kann nun in einem einzigen Step ausgeführt werden.

```
//GO        EXEC PGM=EIGEN,REGION=60K
```

REGION ist nur in MVT-Systemen nötig, damit der für das Programm erforderliche Speicherplatz angefordert wird.

```
//STEPLIB  DD  DSN=PROGRAMM,DISP=SHR
```

Die STEPLIB DD-Karte beschreibt die Datei, die das auszuführende Programm enthält.

```
//FT06F001 DD  SYSOUT=A

//FT07F001 DD  SYSOUT=B

//FT05F001 DD  *

  ***************
  * DATENKARTEN *
  ***************

/*
```

Es soll unterstellt werden, daß SUB1 fehlerhaft ist und deshalb ersetzt werden muß.

```
//STEP     EXEC FORTCLG
//FORT.SYSIN DD  *
  ************************
  * SUB1 QUELLENPROGRAMM *
  ************************
/*
//LKED.SYSLMOD DD  DSN=PROGRAMM(EIGEN),DISP=OLD,SPACE=
```

Wieder wird SYSLMOD mit der Datei, die das neue Lademodul enthalten soll, überschreiben. Da EIGEN bereits Glied in der Datei PROGRAMM ist, wird es durch das neue Lademodul EIGEN ersetzt. Der Parameter SPACE= muß kodiert werden, damit der SPACE-Parameter in der überschriebenen SYSLMOD Karte gelöscht wird. Auf diese Art und Weise wird die Sekundär-Zuordnung, die bei der Erzeugung von PROGRAMM angegeben wurde, nicht geändert.

```
//LKED.SYSIN DD  *
 bENTRY   MAIN
```

Die Routine MAIN wird nun nicht mehr zuerst geladen, und deshalb muß dem Linkage Editor mitgeteilt werden, daß MAIN Eingangspunkt ist.

```
 bINCLUDE   SYSLMOD
```

Das alte Lademodul EIGEN wird als zusätzliche Eingabe hinzugenommen.

```
/*
//GO.SYSIN DD  *
  ***************
  * DATENKARTEN *
  ***************
/*
```

13.1.4 Erzeugung von Unterprogrammbibliotheken

Beim Linkage Editor-Lauf können auch private Unterprogrammbibliotheken erzeugt werden. Die NAME-Karte grenzt ein Lademodul ab, gibt

ihm einen Namen und erlaubt es, mehrere Lademodule in einem einzigen Linkage Editor-Step auszugeben. Die NAME-Karte muß hinter das letzte, in das Lademodul aufzunehmende Objektmodul gelegt werden. ALIAS-Karten müssen vor der NAME-Karte liegen. Die NAME-Karte wird wie folgt kodiert:

```
bNAME   name
     oder
bNAME   name(R)
```

Für das Lademodul kann jeder beliebige Name ausgewählt werden. Da dieser Name aber beim Durchsuchen der Bibliothek verwendet wird, sollte der Name des Unterprogramms verwendet werden. Man kodiert (R), falls das Unterprogramm ein in der Bibliothek bestehendes Unterprogramm ersetzen soll. (Wird (R) für ein Modul bei einer Ersetzung weggelassen, so wird das neue Modul unter dem Namen TEMPNAME hinzugefügt.)

```
bNAME   SUB1                    SUB1 wird hinzugefügt
bNAME   SUB2(R)                 SUB2 wird ersetzt
```

Nehmen wir an, es soll eine private Unterprogrammbibliothek namens MYLIB mit den Gliedern SUB1 und SUB2 erzeugt werden. An sich sollte man ein aus dem Linkage Editor allein bestehendes Verfahren verwenden. Da wir hier aber kein solches haben, modifizieren wir das Verfahren FORTCLG.

```
//JOBNAME  JOB (2211,60),MUELLER,MSGLEVEL=1,RESTART=STEP1.LKED
```

Der Parameter RESTART kann dazu verwendet werden, um mit einem anderen als dem ersten Jobstep zu starten.

```
//STEP1    EXEC FORTCLG,PARM.LKED='LIST,NCAL,LET',COND.GO=ONLY
```

Es werden die Optionen NCAL und LET verwendet, damit jedes Lademodul nur ein einziges Unterprogramm enthält. Der GO-Step wird übersprungen, indem man seine Ausführung davon abhängig macht, ob einer der vorausgegangenen Steps abnormal endet - ein ganz unwahrscheinlicher Fall.

```
//LKED.SYSLMOD DD  DSN=MYLIB,DISP=(,CATLG),      -
//                 SPACE=(1024,(10,5,10))
```

SYSLMOD wird überschrieben, um die Datei MYLIB anzulegen.

```
//LKED.SYSLIN DD  *
```

Die erste DD-Karte für SYSLIN muß überschrieben werden, weil diese Datei nicht vom vorausgehenden Step übergeben wird.

```
bINCLUDE   DD1(SUB1)
```

Mit dieser Karte wird SUB1 verarbeitet. SUB1 ist Glied der in der DD1 DD-Karte beschriebenen Datei.

```
bNAME   SUB1
```

Das SUB1 enthaltende Lademodul wird wieder SUB1 genannt.

```
bINCLUDE   DD1(SUB2)
```

Mit dieser Karte wird SUB2 verarbeitet.

```
bNAME    SUB2
```

Das SUB2 enthaltende Lademodul wird wieder SUB2 genannt.

```
/*
//LKED.DD1 DD  DSN=PROGRAMM,DISP=SHR
```

Diese DD1 DD-Karte beschreibt die Datei PROGRAMM, die SUB1 und SUB2 enthält.

MYLIB enthält nun die zwei Unterprogramme. Die Bibliothek läßt sich mit der SYSLIB DD-Karte verknüpfen und als Unterprogrammbibliothek verwenden.

```
//STEP     EXEC FORTCLG
//FORT.SYSIN DD  *
  ****************************
  * FORTRAN-QUELLENPROGRAMME *
  ****************************
/*
//LKED.SYSLIB DD
```

Man benutzt eine DD-Karte mit leerem Operandenfeld dazu, um in einem katalogisierten Verfahren über eine DD-Karte hinwegzusetzen.

```
//          DD  DSN=MYLIB,DISP=SHR
```

Die Datei MYLIB wird mit der Datei SYS1.FORTLIB verknüpft.

```
//GO.SYSIN DD  *
  ****************
  * DATENKARTEN *
  ****************
/*
```

Jetzt bleibt nur noch übrig, die Ersetzung eines Unterprogramms in einer Bibliothek zu zeigen. Der folgende Job ersetzt SUB1.

```
//STEP1    EXEC FORTCLG,PARM.LKED=(LIST,NCAL,LET),COND.GO=ONLY
```

Es wird nur der Compile- und Link-Step verwendet, der GO-Step wird übergangen.

```
//FORT.SYSIN DD  *
  ***********************
  * SUB1 QUELLENPAKET *
  ***********************
/*
//LKED.SYSLMOD DD  DSN=MYLIB,DISP=OLD,SPACE=
//LKED.SYSIN DD  *
bNAME  SUB1(R)
```

Die NAME-Karte folgt der primären Eingabe, nämlich dem Objektmodul SUB1.

```
/*
```

Mit dem obigen Job kann zu einer Zeit immer nur ein übersetztes Unterprogramm in der MYLIB ausgetauscht werden. Das liegt daran, daß es unmöglich ist, NAME-Karten zwischen die von einem Compiler erzeugten Objektmodule einzufügen. Sollen mehrere Unterprogramme ausgetauscht werden, so müssen sie zuerst übersetzt werden, um Objektmodul-Kartenpakete zu erhalten. Anschließend fügt man die richtigen NAME-Karten hinter die Objektpakete und führt mit diesen Objektmodulen als zweiten Lauf einen Linkage Editor-Lauf durch.

13.1.5 Überlagerungs-Strukturen

Wenn ein Programm zu groß ist um in den Kernspeicher zu passen, kann der Linkage Editor das Programm in mehrere Segmente teilen,

von denen jedes klein genug ist, um in den Kernspeicher zu passen. Jedes Segment wird dann bei Bedarf geladen, indem es das vorige Segment überlagert.

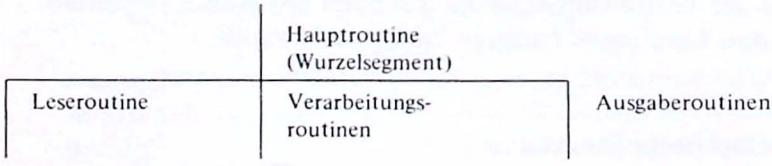
13.1.5.1 *Segmentierung des Programms*

Ein *Segment* ist die kleinste aus einem oder mehreren Programmabschnitten bestehende Programmeinheit, die während der Programmausführung als logische Ganzheit geladen werden kann. Das *Wurzelsegment* (erstes Segment) ist derjenige Teil, der während der gesamten Ausführung des Programms im Kernspeicher bleiben muß. Sollte das ganze Programm während der Ausführung im Kernspeicher bleiben müssen, so kann es nicht segmentiert und überlagert werden.

Ein Programm wird segmentiert, indem man den logischen Ablauf im Programm verfolgt, um die Abhängigkeiten zwischen den Unterprogrammen festzustellen. Man stelle sich beispielsweise vor, daß ein Programm ein Kartenpaket einliest, dann seine Verarbeitung durchführt und endlich die Ergebnisse andruckt. Dieses Programm läßt sich in die folgenden vier Segmente unterteilen:

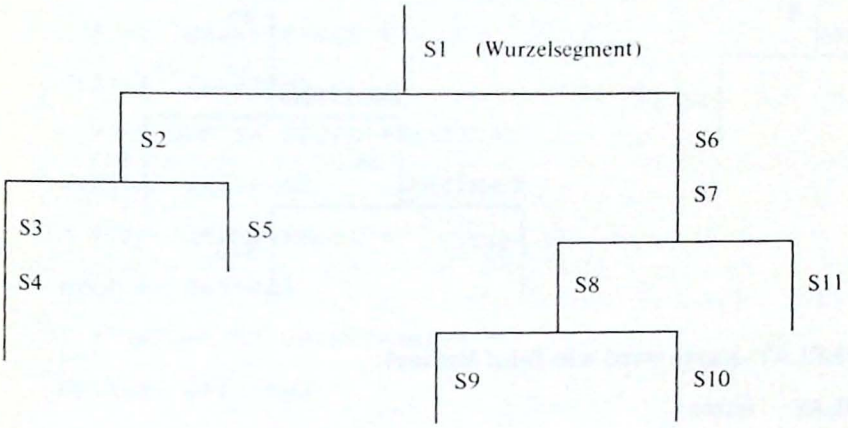
1) Ein Wurzelsegment, das die Hauptroutine des Programms enthält.
2) Ein Segment mit der Routine zum Einlesen des Kartenpakets.
3) Ein Segment mit der Routine, die die Verarbeitung ausführt.
4) Ein Segment, das die Routine zum Andrucken der Ergebnisse enthält.

Das folgende Diagramm heißt *Überlagerungsbaum* [3]) und veranschaulicht die Struktur des Programms. Die Hauptroutine ruft die Leseroutine auf und bewirkt dadurch das Laden des ersten Segmentes. Wenn die Hauptroutine die Verarbeitungsroutinen aufruft, überlagern diese die Leseroutine. Die Ausgaberoutine überlagert die Verarbeitungsroutinen.



[3]) Es sind mehrfache Strukturen möglich, sie gehen aber über den Rahmen dieses Buches hinaus.

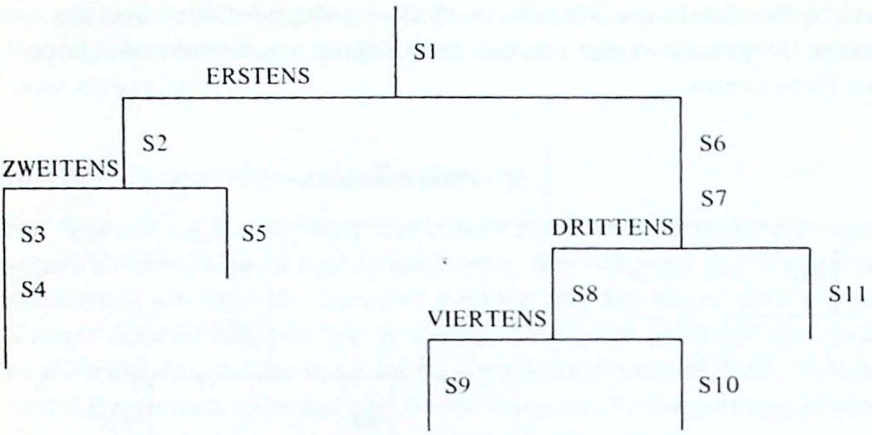
Jede senkrechte Linie im Diagramm stellt ein Segment dar. Das anschließende Diagramm zeigt ein aus neun Segmenten bestehendes komplexeres Programm.



S1 bis S11 stellen Unterroutinen dar (S1 ist die Hauptroutine), und der Überlagerungsbaum zeigt die Segmentierung. Wenn man die Baumstruktur abwärts verfolgt, sagen wir von S1 nach S2 und S5 ohne zurückzugehen, so spricht man von einem *„Weg"*. Segmente entlang eines Wegs können alle zur gleichen Zeit im Speicher sein. Eine Unterroutine darf nur Unterroutinen, die auf ihrem Weg liegen, aufrufen. Das bedeutet, S3 kann S4 aufrufen, nicht aber S5, dagegen kann S2 S3 und S4 oder S5 aufrufen.

13.1.5.2 *Die OVERLAY-Steuerkarte*

Die Steuerkarte OVERLAY des Linkage Editor beschreibt die Überlagerungsstruktur für den Linkage Editor. Man beginnt damit, den Anfang jedes Segmentes zu benennen (das bedeutet, die waagrechten Linien der Baumstruktur erhalten Namen). Die Namen sind beliebig, sie werden aus 1-8 alphanumerischen Zeichen (A-Z, 0-9) gebildet, von denen das erste alphabetisch (A-Z) sein muß. Für unser Beispiel werden die Namen ERSTENS, ZWEITENS, DRITTENS und VIERTENS gewählt.



Die OVERLAY-Karte wird wie folgt kodiert:

bOVERLAY name

Name ist der Name des Segmentanfangs, also ERSTENS, ZWEITENS etc. Die JCL für einen OVERLAY-Lauf lautet wie folgt:

```
//STEP     EXEC VERFAHREN,PARM.LKED=OVLY
```

Das Verfahren soll irgendein katalogisiertes Verfahren mit einem Link-Step sein.

```
.
.
.
//LKED.SYSIN DD  *

  ************************************
  * OBJEKTPAKETE UND OVERLAY-KARTEN  *
  ************************************

/*
```

Der Parameter PARM.LKED=OVLY sagt dem Linkage Editor, daß OVERLAY-Karten folgen. Um die Objektpakete und OVERLAY-Karten in die richtige Reihenfolge zu bringen, verfolgt man die Wege der Baumstruktur abwärts. Jedes Segment wird dabei durch eine OVERLAY-Karte, gefolgt von den richtigen Objektpaketen, beschrieben. Man geht von oben nach unten und von links nach rechts vor. Das Beispiel für die obige Baumstruktur wird diesen einfachen Vorgang verdeutlichen.

```
//LKED.SYSIN DD *
      * S1  OBJEKTPAKET *
    OVERLAY ERSTENS
      * S2  OBJEKTPAKET *
    OVERLAY ZWEITENS
      * S3 UND S4 OBJEKTPAKETE *
    OVERLAY ZWEITENS
      * S5  OBJEKTPAKET *
    OVERLAY ERSTENS
      * S6 UND S7 OBJEKTPAKETE *
    OVERLAY DRITTENS
      * S8  OBJEKTPAKET *
    OVERLAY VIERTENS
      * S9  OBJEKTPAKET *
    OVERLAY VIERTENS
      * S10 OBJEKTPAKET *
    OVERLAY DRITTENS
      * S11 OBJEKTPAKET *
/*
```

Liegen die Objektmodule nicht in Kartenform vor, sondern sind sie in einer Datei auf Plattenspeicher enthalten, so lassen sie sich leicht durch die Linkage Editor-INCLUDE-Karte in die Überlagerungsstruktur einschließen. Man legt die INCLUDE-Karte an die Stelle, an der das Objektpaket liegen würde.

13.1.5.3 *Die INSERT-Steuerkarte*

Es ist oft sehr unbequem, wenn man die Programmabschnitte tatsächlich physisch hinter die betreffenden OVERLAY-Karten legen muß. Statt die Programmabschnitte neu anzuordnen, kann man sie irgendwo in die Primäreingabe legen. Man benutzt dann die INSERT-Karte, um dem Linkage Editor zu sagen, wie die Programmabschnitte angeordnet werden sollen.

Man legt die den Programmabschnitt benennende INSERT-Karte hinter die OVERLAY-Karte, die das Segment definiert, dem dieser Programmabschnitt angehören soll. Gehört der Programmabschnitt aber dem Wurzelsegment an, so wird die INSERT-Karte vor die erste OVERLAY-Karte gelegt. Die INSERT-Karte wird wie folgt kodiert:

bINSERT name,...,name

Die in dieser Karte benannten Programmabschnitte werden an der Stelle eingefügt, an der die INSERT-Karte liegt. Die Programmabschnitte können Objektmodule sein, die von einem vorangegangenen Compile-Step übergeben wurden; es können aber auch Objektpakete sein oder durch INCLUDE-Karten eingebrachte Objekt- und Lademodule. Die Programmabschnitte selbst können irgendwo in der Primäreingabe liegen, vor oder hinter der INSERT-Karte. Werden die in der INSERT-Karte genannten Programmabschnitte in der Primäreingabe nicht gefunden, so wird die Einrichtung automatischer Bibliotheksaufruf eingesetzt.

Das folgende Beispiel veranschaulicht die Verwendung der INSERT- und INCLUDE-Karten bei der Segmentierung des Programms aus dem vorangegangenen Beispiel. Es soll dabei unterstellt werden, daß S6 in der sequentiellen Datei SUBS enthalten ist, daß S3 und S8 vom Compile-Step herübergereicht wurden und daß S10 und S11 in der untergliederten Datei MORSUBS enthalten sind.

```
//STEP     EXEC FORTCLG,PARM.LKED=OVLY
```

Es wird das FORTRAN Compile-Link-Go-Verfahren aufgerufen.

```
//FORT.SYSIN DD  *

  **********************
  * S3 QUELLENPROGRAMM *
  * S8 QUELLENPROGRAMM *
  **********************

/*

//LKED.A   DD  DSN=SUBS,DISP=SHR
```

Die DD-Karte definiert SUBS.

```
//LKED.B   DD  DSN=MORSUBS,DISP=SHR
```

Auch MORSUBS wird definiert.

```
//LKED.SYSIN DD  *
  ****************************************
  * S1, S2, S4, S5, S7, S9 OBJEKTPAKETE *
  ****************************************
```

Die Objektpakete können irgendwo liegen.

```
   INCLUDE B(S10,S11)          ANDERE HINZUFUEGUNGEN
   INSERT  S1                  FUEGE S1 HIER EIN
   OVERLAY ERSTENS
   INSERT  S2                  FUEGE S2 HIER EIN
   OVERLAY ZWEITENS
   INSERT  S3,S4               FUEGE S3 UND S4 HIER EIN
   OVERLAY ZWEITENS
   INSERT  S5                  FUEGE S5 HIER EIN
   OVERLAY ERSTENS
   INCLUDE A                   HINZUFUEGUNG VON SUBS
   INSERT  S7                  FUEGE S7 HIER EIN
   OVERLAY DRITTENS
   INSERT  S8                  FUEGE S8 HIER EIN
   OVERLAY VIERTENS
   INSERT  S9                  FUEGE S9 HIER EIN
   OVERLAY VIERTENS
   INSERT  S10                 FUEGE S10 HIER EIN
   OVERLAY DRITTENS
   INSERT  S11                 FUEGE S11 HIER EIN
/*
```

13.2 Verdichtung von Dateien

Untergliederte Dateien werden größer, wenn man ihnen neue Glieder hinzufügt oder wenn man alte Glieder ersetzt. Beide nämlich, die

neuen Glieder und die als Ersetzung eingegebenen Glieder werden am Ende der Datei angefügt. Der Name des Gliedes sowie die Stelle, an die es gespeichert wurde, werden in das Inhaltsverzeichnis (Directory) der untergliederten Datei gestellt. Es ist wichtig, sich zu vergegenwärtigen, daß ein zur Ersetzung eingegebenes Glied nicht den Platz des zu ersetzenden Gliedes einnimmt. Der von dem alten zu ersetzenden Glied eingenommene Platz ist für eine Neuzuordnung nicht verfügbar, es sei denn, die Datei wird von neuem angelegt oder auf dem alten Platz verdichtet.

Durch das Hinzufügen und Ersetzen von Gliedern kann es vorkommen, daß die Datei voll wird. Das IBM-Dienstprogramm IEBCOPY verdichtet untergliederte Dateien, so daß der von ersetzten Gliedern belegte Platz wieder zur Verfügung steht. (Dateien mit der Folgespureinrichtung können allerdings nicht verdichtet werden.) Das folgende katalogisierte Verfahren zeigt, wie Dateien verdichtet werden.[4])

```
//COMPRESS PROC
//GO       EXEC PGM=IEBCOPY,PARM=COMPRESS,REGION=20K
```

IEBCOPY ist in der SYS1.LINKLIB enthalten. PARM=COMPRESS fordert die Verdichtungsoperation an. REGION ist nur in MVT-Systemen nötig.

```
//SYSPRINT DD  SYSOUT=A
```

SYSPRINT ist die Druckdatei für das Protokoll von IEBCOPY.

```
//SYSUT1   DD  DSN=&NAME,DISP=OLD
```

SYSUT1 beschreibt die zu verdichtende Datei. Der Dateiname enthält einen symbolischen Parameter, den der Anwender des Verfahrens in der EXEC-Karte kodieren muß. Wenn die Datei nicht katalogisiert ist, muß der Anwender die Karte SYSUT1 überschreiben, um die Parameter UNIT und VOL hinzuzufügen.

```
//SYSUT2   DD  DSN=*.SYSUT1,DISP=OLD,VOL=REF=*.SYSUT1
```

SYSUT2 beschreibt die Ausgabedatei; es ist die gleiche wie die von SYSUT1 beschriebene. (Der Parameter VOL wurde für den Fall hinzugefügt, daß SYSUT1 keine katalogisierte Datei beschreibt.)

---

[4]) In diesem und dem nächsten Kapitel werden im Zusammenhang mit katalogisierten Verfahren verschiedene Beispiele gebracht, die dem Leser beim Schreiben seiner eigenen Verfahren helfen sollen.

```
//SYSIN     DD  DUMMY
```

SYSIN ist nicht erforderlich; sie wird deshalb zur Pseudodatei erklärt.

Wir nehmen jetzt an, daß die Datei MAPS verdichtet werden soll. Wenn die Datei katalogisiert ist, besteht der Step nur aus einer Karte.

```
//STEPCOMP EXEC COMPRESS,NAME=MAPS
```

Wenn die Datei MAPS dagegen nicht katalogisiert ist, muß die Karte SYSUT1 überschrieben werden, um die Informationen UNIT und VOL zur Verfügung zu stellen.

```
//STEPCOMP EXEC COMPRESS,NAME=MAPS
//GO.SYSUT1 DD  UNIT=2311,VOL=SER=DA0300
```

Hat eine Datei ihre Primär-Zuordnung und ihre Sekundär-Zuordnungen oder ihren Platz im Directory aufgebraucht, dann hilft kein Verdichten mehr. Die Datei muß neu angelegt werden. Das kann auch mit IEBCOPY geschehen.

```
//ANLEGEN  PROC
//GO       EXEC PGM=IEBCOPY,REGION=20K
//SYSPRINT DD  SYSOUT=A
//SYSUT1   DD  DSN=&NAME,DISP=(OLD,DELETE)
```

SYSUT1 beschreibt die alte Datei. Sie wird gelöscht, weil eine neue Datei angelegt wird.

```
//SYSUT2   DD  DSN=*.SYSUT1,DISP=(,CATLG),
//            UNIT=(SYSDA,SEP=SYSUT1),SPACE=&SPACE,DCB=*.SYSUT1
```

Die neue Datei muß auf einem anderen Datenträger angelegt werden, weil der gleiche Name beibehalten werden soll. Zur Vereinfachung wurde der Wert für SPACE als symbolischer Parameter kodiert.

```
//SYSIN     DD  DUMMY
```

Die Datei MAPS kann nun mit mehr Platz neu angelegt werden.

```
//STEPNEU  EXEC ANLEGEN,NAME=MAPS,SPACE='(1024,(30,20,50))'
```

13.3 Zusammenfassung

Der Linkage Editor dient dazu, Programme zur Ausführung vorzubereiten, sie als Glieder untergliederter Dateien aufzubewahren und Unterprogrammbibliotheken zu erzeugen. All dies wird erreicht durch eine Verbindung von JCL- und Linkage Editor-Kontrollkarten. Programme, die zu groß sind, um in den Kernspeicher zu passen, können segmentiert werden, so daß die verschiedenen Segmente einander überlagern.

Das Dienstprogramm IEBCOPY verdichtet untergliederte Dateien, um den von alten Gliedern besetzten Platz wieder zur Verfügung zu stellen. Sollte mehr Platz erforderlich sein, kann IEBCOPY die Datei mit mehr Platz neu erzeugen.

Die vom PARM-Feld der EXEC-Karte angegebenen und häufig benutzten Linkage Editor-Parameter sind:

LIST	Druckt die Linkage Editor-Kontrollkarten an.
MAP oder XREF	Erzeugt einen Speicherbelegungsplan, der die Lage der Programmabschnitte zeigt. Zusätzlich dazu erzeugt XREF eine Zuordnungstabelle.
NCAL	Schaltet den automatischen Bibliotheksaufruf aus.
LET	Kennzeichnet Lademodule als ausführbar, selbst wenn kleine Fehler gefunden wurden.
HIAR	Gestattet es, Programmabschnitte in den Hierarchie-Speicher zu legen.
OVLY	Erlaubt es, Programmabschnitte für die Überlagerung zu segmentieren.

Die gebräuchlichsten Linkage Editor-Kontrollkarten sind:

bENTRY name
Benennt den Eingangspunkt.

bALIAS name,...
Ordnet dem Lademodul Zweitnamen zu.

bINCLUDE ddname,...
Stellt sequentielle Dateien als Linkage Editor-Eingabe zur Verfügung.

bINCLUDE ddname(member,...),...
Stellt Glieder untergliederter Dateien als Linkage Editor-Eingabe zur Verfügung.

bLIBRARY ddname(member,...),...
Führt die Namen von Gliedern auf, nach denen in einer Bibliothek gesucht werden soll.

bLIBRARY (name,...)
Gestattet es, für einen Lauf externe Referenzen ungelöst zu lassen.

bLIBRARY *(name,...)
Gestattet es, externe Referenzen für die Lebensdauer des Lademoduls ungelöst zu lassen.

bHIARCHY 1,name,...
Legt Programmabschnitte in den Hierarchie-Speicher.

bNAME name
Benennt ein hinzuzufügendes Lademodul.

bNAME name(R)
Benennt ein Lademodul, das ersetzt werden soll.

bOVERLAY name
Benennt ein Segment für die Überlagerungsstruktur.

bINSERT name,...,name
Fügt Programmabschnitte in eine Überlagerungsstruktur ein.

14. Verschiedene JCL-Einrichtungen

Dieses Kapitel führt die übrig gebliebenen JCL-Einrichtungen auf: Die Befehls-, Null- und Kommentar-Karte, die Pflege der Datei, Generation Data Groups und Dateien über mehrere Datenträger. Es wird der Dateischutz beschrieben sowie mehrere Punkte, die die E/A-Leistungsfähigkeit betreffen. Nach alter Tradition wird das Schwerste bis zum Schluß aufgehoben und so werden schließlich die direkten und index-sequentiellen Dateien beschrieben.

14.1 Befehls-Anweisung

Die Operator-Befehle werden normalerweise vom Konsol her eingegeben. Jedoch können häufig benutzte Befehle auf Befehls-Anweisungen kodiert werden, um mit dem Eingabestrom eingelesen zu werden. Befehls-Anweisungen sollten nur vom Operator benutzt werden, weil sie die Arbeit des Systems stark beeinflussen.

Die Befehle müssen mit der Ausführung des Jobs synchronisiert werden, weil sie nämlich, sobald sie eingelesen sind, auch ausgeführt werden (eine Ausnahme bilden die Befehle SET und START in PCP-Systemen). Die Befehls-Anweisung wird vor eine JOB-, EXEC-, Null- oder andere Befehls-Anweisung gelegt. Sie darf mit einer Folgekarte nicht fortgesetzt werden. Befehls-Anweisungen werden wie folgt kodiert:

```
// befehl operanden
```

Der Befehl ist ein Operator-Befehl; seine Operanden hängen vom Befehlstyp ab. Die folgenden Befehle können als Befehls-Anweisungen in die verschiedenen Systeme eingegeben werden.

PCP:	DISPLAY	SET	STOP	VARY
	MOUNT	START	UNLOAD	

MFT:	CANCEL	LOG	REPLY	STOP
	DISPLAY	MODIFY	RESET	UNLOAD
	HALT	MOUNT	SET	VARY
	HOLD	RELEASE	START	WRITELOG
MVT:	CANCEL	LOG	REPLY	STOP
	DISPLAY	MODIFY	RESET	UNLOAD
	HALT	MOUNT	SET	VARY
	HOLD	RELEASE	START	WRITELOG

Beispiele

```
// VARY 293,OFFLINE
// DISPLAY Q
```

14.2 Null-Anweisung

Die Null-Karte enthält in den Spalten 1 und 2 // und ist in den restlichen Stellen leer. Diese Karte veranlaßt das System nach der nächsten JOB-Karte zu suchen. Alle Karten zwischen ihr und der nächsten JOB-Karte werden ignoriert. Manchmal werden Null-Anweisungen auf farbigen Karten benutzt, um dem Operator beim Trennen der Jobs zu helfen. Der Operator benutzt die Null-Karte auch um das Drücken der Taste "Letzte Karte" am Leser zu vermeiden. Der Reader müßte enden, um die letzte Karte einzulesen. Ein Operator Befehl wäre notwendig, um ihn wieder anlaufen zu lassen. Der Operator kann hinter den letzten Job 4 Null-Karten in den Kartenleser legen, um zu erreichen, daß der Job vollständig eingelesen wird, ohne daß der Reader endet.

14.3 Kommentar -Anweisung

Die Kommentar-Anweisung enthält in den Spalten 1 bis 3 //*. Die restlichen Spalten können jeden beliebigen Kommentar enthalten. Sie können an beliebiger Stelle hinter der JOB-Karte liegen, und dienen zur Dokumentation des Jobs. Die Kommentar-Karten werden in der System Ausgabe mit *** in den Spalten 1 bis 3 aufgelistet. Kommentar-Karten dürfen keine Folgekarten haben; sollte der Kommentar länger als eine

Karte sein, kann der Anwender in der folgenden Karte wieder //* kodieren und den Kommentar auf ihr fortsetzen. Kommentar darf auch auf jeder anderen JCL-Karte kodiert werden, indem eine Leerstelle hinter dem letzten Operanden gelassen wird.

```
//A        DD  UNIT=2311,     EINE 2311 PLATTE WIRD BENUTZT
//*  DIE FOLGENDE KARTE FORDERT
//*  EINEN BESTIMMTEN DATENTRAEGER AN.
//              VOL=SER=DA0200  PLATTE DA0200 WIRD BENUTZT
```

Das System listet den obigen Kommentar mit an, ignoriert ihn aber beim Interpretieren der JCL. Die JCL wird wie folgt interpretiert.

```
//A        DD  UNIT=2311,VOL=SER=DA0200
```

14.4 Pflege der Dateien

14.4.1 Katalogisierte Dateien

Wird eine Datei katalogisiert, so werden ihr Name (DSN), ihre Einheit (UNIT) und die Nummer ihres Datenträgers bzw. die Nummern ihrer Datenträger (VOL=SER) in die Systemdatei SYSCTLG eingetragen. Für Banddateien wird auch die Dateifolgenummer mit aufgezeichnet. Später kann man sich auf diese katalogisierte Datei beziehen, ohne daß ihre Einheit, ihre Datenträgernummer und ihre Dateifolgenummer angegeben werden muß.

Meist ist der Katalog vollständig auf der Systemresidenz enthalten. Er kann jedoch auch auf andere Platten ausgedehnt werden, um auf der Systemresidenz Platz zu sparen. Oft wird der Katalog auf die Platte ausgedehnt, die die von ihm beschriebenen Dateien enthält. Auf diese Weise lassen sich die Dateien und der dazugehörige Katalog leicht von System zu System transportieren. Jede Platte, die einen Teil des Katalogs enthält, heißt *Control Volume.*

Das IBM-Dienstprogramm IEHPROGM erzeugt solche Control Volumes, indem es einen Indexeintrag in einem bestehenden Katalog vornimmt, der auf den Control Volume verweist. Da das Programm IEHPROGM anderen wichtigen Anwendungen dient, sollte der Leser mit ihm vertraut sein.

14.4.2 Das Dienstprogramm IEHPROGM

Das IBM-Dienstprogramm IEHPROGM dient dazu, Dateien zu löschen (SCRATCH), es löscht Glieder untergliederter Dateien, es katalogisiert und entkatalogisiert Dateien, und es benennt Dateien oder Glieder untergliederter Dateien um. Das Programm baut Indices auf und löscht sie wieder, es richtet Zweitindices (Index Alias) und Indices für Generation Data Groups ein. (IEHPROGM hat noch andere hier nicht aufgeführte Anwendungen für die Systempflege.) IEHPROGM wird wie folgt ausgeführt.

```
//PFLEGE   PROC
//GO       EXEC PGM=IEHPROGM,REGION=48K
```

IEHPROGM ist in der SYS1.LINKLIB enthalten. Der Parameter REGION ist nur bei MVT-Systemen nötig.

```
//SYSPRINT DD   SYSOUT=A
```

SYSPRINT ist die Druckausgabe für das IEHPROGM Protokoll.

```
//SYSTEM   DD   DSN=SYS1.SVCLIB,DISP=OLD
```

Diese DD-Karte definiert die Systemresidenz. Der ddname ist beliebig, darf aber nur einmal vorkommen.

//DDn DD UNIT=*gerät*, VOL=SER=*datenträgernummer*, DISP=OLD

Für jeden Datenträger, der auf den weiter unten beschriebenen Datenkarten im VOL- oder CVOL-Parameter genannt wird, muß eine solche DD-Karte mit einem eindeutigen ddname zur Verfügung gestellt werden. Liegen diese DD-Karten nicht in dem Verfahren, so müssen sie beim Aufruf des Verfahrens hinzugefügt werden. Die Werte gerät und datenträgernummer müssen explizit kodiert werden (etwa 2311, SYSDA und DA0200 etc.).

Das Verfahren wird wie folgt aufgerufen.

```
//AUFRUF   EXEC PFLEGE
//GO.SYSIN DD   *
  ****************
  * DATENKARTEN *
  ****************
/*
```

In einem Lauf kann jede beliebige Anzahl von Datenkarten eingefügt werden. Die Datenkarten bestehen aus einem Operationsfeld, das in Spalte 2 oder später beginnt. Es folgt wenigstens eine Leerstelle und danach das Operandenfeld. Sollten sich Daten über Spalte 71 hinaus erstrecken, so muß die Datenkarte nach einem Komma, aber spätestens in Spalte 71 unterbrochen werden. In Spalte 72 wird ein beliebiges Zeichen gelocht. Die Folgekarte beginnt in Spalte 16. Es können folgende Datenkarten verwendet werden.

1) SCRATCH löscht eine Datei oder das Glied einer untergliederten Datei, die auf einem Speicher mit direktem Zugriff liegen. SCRATCH ist der Disposition DISP=(..,DELETE) ähnlich. Allerdings entkatalogisiert DELETE die Dateien, während SCRATCH das nicht tut. SCRATCH kann auch das Glied einer untergliederten Datei löschen. SCRATCH hat vor DELETE einen besonderen Vorteil: Ein Job, der eine DD-Karte mit der Disposition DELETE enthält, terminiert, wenn die Datei nicht gefunden werden kann. IEHPROGM dagegen geht zur nächsten Datenkarte über.

```
bSCRATCH   VOL=gerät=datenträgernummer,DSNAME=name,
           MEMBER=name,PURGE[1])
```

DSNAME benennt die Datei; der Parameter darf nicht mit DSN abgekürzt werden. VOL gibt das Gerät und die Datenträgernummer des Datenträgers an, auf dem die Datei liegt. Erstreckt sich die Datei über mehrere Datenträger, so kodiert man VOL=gerät=(datenträgernummer,...,datenträgernummer). MEMBER gibt das Glied der untergliederten Datei an, das gelöscht werden soll. Soll die ganze untergliederte Datei gelöscht werden, so läßt man MEMBER weg. PURGE ist für Dateien mit Schutzdatum.

```
bSCRATCH   VOL=2314=DA0300,DSNAME=DATEI7,PURGE
```

Die Datei DATEI7 auf dem Plattenstapel DA0300 einer 2314-Plattenspeichereinheit wird gelöscht. PURGE schaltet das Schutzdatum aus.

```
bSCRATCH   VOL=2311=DA0400,DSNAME=TESTLIB,
           MEMBER=EINS
```

Das Glied EINS der Datei TESTLIB, die auf dem 2311-Plattenstapel DA0400 liegt, wird gelöscht.

---

[1]) Das "b" steht für eine Leerstelle und soll daran erinnern, daß die Karte nicht in Spalte 1 anfangen darf.

2) RENAME gestattet es, die Namen oder Zweitnamen von Dateien oder von Gliedern untergliederter Dateien, die auf Speichern mit direktem Zugriff liegen, zu ändern. RENAME verändert den Katalogeintrag nicht. Soll der Katalogeintrag berichtigt werden, so müssen UNCATLG und CATLG verwendet werden.

```
bRENAME   VOL=gerät=datenträgernummer,DSNAME=name,
          NEWNAME=name,MEMBER=name
```

Der Parameter DSNAME gibt den gegenwärtigen Namen der Datei an. VOL stellt das Gerät und die Datenträgernummer des Datenträgers, auf dem die Datei liegt, zur Verfügung. Der Parameter NEWNAME benennt den neuen Namen der Datei oder des Gliedes. Dreht es sich um ein Glied, so muß mit MEMBER der alte Name des Gliedes angegeben werden. Man läßt MEMBER weg, wenn die ganze Datei umbenannt werden soll.

```
bRENAME   VOL=2311=DA0400,DSNAME=DATEIG,
          NEWNAME=TEST9
```

Die Datei DATEIG wird in TEST9 umbenannt.

```
bRENAME   VOL=2311=DA0400,DSNAME=LIB,NEWNAME=SQRT,
          MEMBER=SQQT
```

Das Glied SQQT der Datei LIB wird in SQRT umbenannt.

3) CATLG katalogisiert eine Datei. Falls nötig, werden höhere Indices dabei mit aufgebaut. Wird beispielsweise A.B.C katalogisiert, so werden die Indices für A und A.B mit aufgebaut. Das ist bei der Disposition DISP=(..,CATLG) anders. Hier können keine höheren Indices aufgebaut werden.

```
bCATLG    VOL=gerät=datenträgernummer,CVOL=gerät=daten-
          trägernummer, DSNAME=name
```

DSNAME benennt die Datei. Der Parameter VOL gibt das Gerät und die Datenträgernummer der zu katalogisierenden Datei an. Falls die Datei sich über mehrere Datenträger erstreckt, muß VOL=gerät=(datenträgernummer, ..., datenträgernummer) kodiert werden. Die Datenträgernummern müssen in der Reihenfolge aufgeführt werden, die vorgelegen hat, als die Datei erzeugt wurde.

Wenn sich die Datei auf einem Magnetband befindet, muß die Dateifolgenummer mit angegeben werden. Man kodiert VOL=gerät=

(datenträgernummer,dateifolgenummer,...,datenträgernummer, dateifolgenummer). Der Parameter CVOL ist sehr selten erforderlich. Er gibt das Gerät und die Datenträgernummer des Datenträgers an, auf dem mit der Suche im Katalog begonnen werden soll. Dieser Suchvorgang beginnt auf der Systemresidenz, falls CVOL fehlt. Man benutzt CVOL, wenn eine Datei in einem Katalog, der nicht zum System gehört, katalogisiert werden soll.

```
bCATLG   VOL=2400=(BD0200,1),DSNAME=DATEI3
```

Die Datei DATEI3, die erste Datei auf dem Band BD0200, wird katalogisiert.

```
bCATLG   VOL=2314=DA0300,DSNAME=DATEI9
```

Die Datei DATEI9, die auf dem 2314-Plattenstapel DA0300 liegt, wird katalogisiert.

4) UNCATLG entkatalogisiert Dateien, ohne daß höhere Indices entfernt werden. Wird beispielsweise A.B.C entkatalogisiert, so bleiben A und A.B als Indices bestehen. UNCATLG ist mit der Disposition DISP=(..,UNCATLG) gleichwertig.

```
bUNCATLG   DSNAME=name,CVOL=gerät=datenträgernummer
bUNCATLG   DSNAME=TAX.STATE
```

Die Datei TAX.STATE wird entkatalogisiert. Der Index bleibt erhalten.

5) BLDX baut einen Index für Dateinamen auf. Man muß BLDX oder CATLG verwenden, bevor Dateien mit qualifizierten Namen katalogisiert werden können. Beispielsweise muß der Index für TAX aufgebaut sein, wenn in der DD-Karte DSN=TAX.STATE,DISP=(,CATLG) kodiert wird. BLDX baut, falls nötig, auch alle höheren Indices auf. D.h., daß beim Aufbau des Index für A.B.C auch die Indices für A und A.B aufgebaut werden, falls das notwendig sein sollte. Der Benutzer kann danach Dateinamen, wie A.name, A.B.name und A.B.C.name benutzen.

```
bBLDX   INDEX=index,CVOL=gerät=datenträgernummer
bBLDX   INDEX=ABT.DATEN
```

Der Anwender kann jetzt Dateinamen von der Form ABT.DATEN.name katalogisieren.

6) DLTX löscht die niedrigste Stufe des Index. Auf diese Weise wird durch das Löschen von A.B.C der Indexeintrag für A und A.B

nicht zerstört. Der Index wird allerdings nicht gelöscht, solange er noch Eintragungen hat. Beispielsweise wird der Index C nicht gelöscht, wenn eine Datei A.B.C.D katalogisiert ist, und man das Löschen für A.B.C versucht.

```
bDLTX   INDEX=index,CVOL=gerät=datenträgernummer
bDLTX   INDEX=ABT.DATEN
```

Der Index für ABT.DATEN wird entfernt. Der Index für ABT bleibt erhalten.

7) BLDA ordnet dem Index der höchsten Stufe einen Zweitnamen zu. Wenn beispielsweise X Zweitname von Z ist, kann man sich auf eine Datei entweder mit Z.B oder mit X.B beziehen.

```
bBLDA   INDEX=name,ALIAS=zweitname,CVOL=gerät=daten-
        trägernummer
bBLDA   INDEX=CITY,ALIAS=TOWN
```

TOWN wird zum Zweitnamen für den Index CITY.

8) DLTA löscht einen mit BLDA erzeugten Zweitnamen.

```
bDLTA   ALIAS=zweitname,CVOL=gerät=datenträgernummer
bDLTA   ALIAS=TOWN
```

Der Zweitname TOWN wird gelöscht.

9) CONNECT verknüpft die Kataloge zweier Datenträger (Control Volumes), indem in dem höher geordneten Katalog ein Eintrag vorgenommen wird, der auf den nachgeordneten Control Volume verweist. Dieser Eintrag befindet sich in der höchsten Stufe des Katalogs. Der Benutzer muß auf dem nachgeordneten Control Volume natürlich auch eine Datei SYSCTLG (eben den Katalog) einrichten und in ihr einen Index anlegen (mit BLDX).

```
bCONNECT   INDEX=index,VOL=gerät=datenträgernummer,
           CVOL=gerät=datenträgernummer
```

INDEX gibt den Namen des zu erzeugenden Indexeintrags an, während VOL die Datenträgernummer des nachgeordneten Control Volume spezifiziert. Der Parameter CVOL ist nur nötig, wenn der neue Control Volume nicht an die Systemresidenz angehängt werden soll. CVOL gibt in diesem Fall den Datenträger an, an den der neue Control Volume angehängt werden soll. Es können mehrere Datenträger aneinandergeknüpft werden.

bCONNECT INDEX=JONES,VOL=2311=DA0300

Der Datenträger DA0300 wird zum Control Volume gemacht. Auf ihm muß eine Datei SYSCTLG eingerichtet werden, in der ein Index für JONES erzeugt wird. Mit CONNECT wird anschließend im Katalog der Systemresidenz ein Eintrag erzeugt, der auf diesen Control Volume verweist. Bezieht sich der Anwender nun auf eine Datei JONES.name, so verweist der Eintrag in der Systemresidenz auf den Datenträger DA0300, und die Suche nach der Datei JONES.name wird im SYSCTLG von DA0300 fortgesetzt.

10) RELEASE löst die mit CONNECT geschaffene Verbindung zwischen zwei Datenträgern wieder auf, indem der Eintrag im höher geordneten Katalog gelöscht wird. Die Indexeintragung im nachgeordneten Katalog bleibt davon unberührt.

bRELEASE INDEX=index,CVOL=gerät=datenträgernummer
bRELEASE INDEX=JONES

Der Indexeintrag für JONES wird im Katalog der Systemresidenz gelöscht. Dadurch wird der Datenträger DA0300 abgetrennt.

11) BLDG baut einen Index für Generation Data Groups auf. Ein solcher muß eingerichtet worden sein, bevor eine Generation Data Group erzeugt werden kann. (Mit DLTX kann ein solcher Index wieder gelöscht werden.)

bBLDG INDEX=name,ENTRIES=anzahl,DELETE,EMPTY,
CVOL=gerät=dateiträgernummer

INDEX gibt den qualifizierten Namen der Generation Data Group an. ENTRIES spezifiziert die Anzahl der zu unterhaltenden Generationen (maximal 255). Die Angabe DELETE bewirkt die Löschung alter Generationen, sobald sie aus dem Index herausfallen. Die Angabe EMPTY bewirkt das Entfernen aller Generationen aus dem Index, sobald eine Generation erzeugt wird, die die Anzahl der in ENTRIES angegebenen Generationen übertrifft. Wird EMPTY nicht kodiert, so wird nur die älteste Generation aus dem Index entfernt.

bBLDG INDEX=APOLLO.ACHT,ENTRIES=2,DELETE

Für APOLLO.ACHT wird ein Index aufgebaut, der 2 Generationen aufnehmen kann. Sobald eine alte Generation aus dem Index entfernt wird, wird sie auch gelöscht.

14.4.2.1 *Beispiele*

Im folgenden Beispiel wird die katalogisierte Datei DAT7, die auf dem Datenträger DA0200 liegt, in EDATEI3 umbenannt. Weil die Datei katalogisiert ist, muß sie auch entkatalogisiert werden, um unter ihrem neuen Namen wieder katalogisiert zu werden.

```
//NAMENEU  EXEC PFLEGE
//GO.A     DD  UNIT=2314,VOL=SER=DA0200,DISP=OLD
```

Die DD-Karte A definiert den Plattenstapel DA0200.

```
//GO.SYSIN DD  *
  RENAME   DSNAME=DAT7,VOL=2314=DA0200,NEWNAME=EDATEI3
  UNCATLG  DSNAME=DAT7
  CATLG    DSNAME=EDATEI3,VOL=2314=DA0200
/*
```

Das nächste Beispiel erzeugt im Katalog der Systemresidenz einen Eintrag für STATE, der auf den Control Volume DA0500 verweist. Anschließend wird die Datei STATE.UNION auf dem neuen Control Volume katalogisiert.

```
//STEPX    EXEC PFLEGE
//GO.A     DD  DSN=SYSCTLG,DISP=(,KEEP),UNIT=2314,
//             VOL=SER=DA0500,SPACE=(TRK,(10,1))
```

Die DD-Karte A erzeugt auf dem Datenträger DA0500 einen Katalog. Die Primär-Zuordnung für ihn beträgt 10 Spuren. Als Sekundär-Zuordnung sind Zuschläge von einer Spur gestattet.

```
//GO.SYSIN  DD  *
  CONNECT      INDEX=STATE,VOL=2314=DA0500
  CATLG        DSNAME=STATE.UNION,VOL=2314=DA0500
/*
```

14.5 Generation Data Groups

Eine Generation Data Group ist eine Gruppe von Dateien, die zueinander in einem zeitlichen oder funktionalen Zusammenhang stehen. Sie werden periodisch verarbeitet. Das geschieht meist so, daß eine neue Generation unter Beibehaltung der früheren Generationen hinzugefügt wird, und daß dafür die älteste Generation ausgeschieden wird. Beispielsweise ist der Einkommenssteuerbescheid eine Generation Data Group. Jedes Jahr wird eine neue Generation hinzugefügt, die mit den Generationen der vorangegangenen Jahre in einem zeitlichen und funktionalen Zusammenhang steht. Wenn eine neue Generation hinzugefügt wird, müssen die Bescheide der vier vorangegangenen Jahre aus gesetzlichen Gründen aufbewahrt werden. Der Bescheid des fünften Jahres aber braucht nicht mehr aufbewahrt zu werden.

Auf katalogisierte Generation Data Groups bezieht man sich durch ihren Namen und eine relative Nummer. Beispielsweise würde sich DSN=STEUER.BESCHEID(0) auf den laufenden Steuerbescheid beziehen, DSN=STEUER.BESCHEID(-1) wäre der Steuerbescheid des vorangegangenen Jahres etc. Eine neue Generation wird durch ihren Namen DSN=STEUER.BESCHEID(+1) erzeugt.

Der Vorteil der Generation Data Group besteht darin, daß alle Dateien den gleichen Namen haben und daß das System Generationen hinzufügt und löscht. Generation Data Groups können sequentiell, direkt oder untergliedert organisiert sein. Sie können auf jedem beliebigen Datenträger, der für ihre Organisationsform geeignet ist, liegen. Generation Data Groups werden wie jede andere Datei verwendet. Der einzige Unterschied ist die relative Nummer der Generation im Parameter DSN.

Um eine Generation Data Group einzurichten, muß zweierlei getan werden. Mit dem IBM-Dienstprogramm IEHPROGM muß ein Index für die Generation Data Group im Katalog (SYSCTLG) erzeugt werden. Ferner muß eine Datei bzw. Pseudodatei auf dem Datenträger eingerichtet werden, der den Katalog enthält. Der Kennsatz dieser Datei enthält den DCB-Unterparameter für die Generation Data Group.

Als Beispiel soll die Generation Data Group STEUER.BESCHEID eingerichtet werden. Die beiden folgenden Steps erzeugen den Index für die Generation Data Group sowie die Pseudodatei.

```
//STEP1    EXEC PFLEGE
//GO.SYSIN DD   *
   BLDG          INDEX=STEUER.BESCHEID,ENTRIES=5,DELETE
```

INDEX gibt den Namen der Generation Data Group an. ENTRIES legt die Anzahl der Generationen fest und DELETE löscht die älteste Generation, sobald sie aus dem Index entfernt wird.

```
/*
//STEP2    EXEC PGM=IEFBR14
//GDG      DD   DSN=STEUER.BESCHEID,DISP=(,KEEP),SPACE=(TRK,1),
//             VOL=REF=*.STEP1.GO.SYSTEM,
//             DCB=(RECFM=FB,BLKSIZE=800,LRECL=80)
```

Die DD-Karte GDG erzeugt eine Datei für die Generation Data Group, die die notwendigen DCB-Unterparameter enthält. Diese Datei muß auf dem Datenträger liegen, der den Katalog mit dem Indexeintrag für die Generation Data Group enthält. Die Datei kann nicht katalogisiert werden. Falls nötig können im DCB auch DSORG, OPTCD, KEYLEN und RKP kodiert werden.

Jetzt kann eine Generation Data Group erzeugt werden.

```
//STEP1    EXEC PGM=ERZEUGE
```

Wir nehmen an, daß das Programm ERZEUGE eine Datei erzeugt.

```
//GEN      DD   DSN=STEUER.BESCHEID(+1),DISP=(,CATLG),
//             UNIT=2314,SPACE=(80,200)
```

Das System sucht automatisch nach der Datei STEUER.BESCHEID, die die erforderlichen DCB-Unterparameter enthält. Es erzeugt dann die Generation und katalogisiert diese Datei. Als Disposition muß für jede neue Generation CATLG verwendet werden. In der DD-Karte dürfen DCB-Unterparameter kodiert werden, die die der Datei STEUER.BESCHEID überschreiben oder ergänzen.

Die Generation (0) ist stets die laufende Generation, (-1) ist die Generation davor, (-2) ist die zweite Generation vor der laufenden etc. Generation (+1) ist die neue Generation. Das (+1) bewirkt, daß alle Generationen am Ende des Jobs eine Stufe runtergesetzt werden. Die Generation, die vorher (-1) war, ist jetzt (-2). Die Generationen haben wäh-

rend der Dauer des ganzen Jobs die gleiche relative Nummer. Diese Nummern werden nicht verändert, es sei denn, der Job endet.

```
//STEP1    EXEC PGM=EINS
//INPUT    DD  DSN=STEUER.BESCHEID(0),DISP=OLD
```

Diese Datei ist die laufende Generation.

```
//OUTPUT   DD  DSN=STEUER.BESCHEID(+1),DISP=(,CATLG),
//             SPACE=(80,200)
```

Diese DD-Karte erzeugt eine neue Generation.

```
//STEP2    EXEC PGM=ZWEI
//NAECHSTE DD  DSN=STEUER.BESCHEID(+2),DISP=(,CATLG),
//             SPACE=(80,200)
```

Diese DD-Karte erzeugt eine weitere Generation. Es ist nicht möglich, sich auf diese mit (+1) zu beziehen, denn der Generation Data Group-Index wird erst bei Jobende fortgeschrieben. Wenn das geschieht, wird aus (+2) ein (0), aus (+1) ein (-1) und aus (0) ein (-2).

Wenn man der Pseudodatei einen anderen Namen gibt, darf sie auch katalogisiert werden. Sie muß aber nach wie vor auf dem Datenträger liegen, dessen Katalog den Generation Data Group-Index enthält.

```
//STEP2    EXEC PGM=IEFBR14
//GDG      DD  DSN=PSEUDODS,DISP=(,CATLG),SPACE=(TRK,1),
//             UNIT=2314,VOL=SER=SYSRES,
//             DCB=(RECFM=FB,BLKSIZE=800,LRECL=80)
```

Jede Generation Data Group kann die Datei PSEUDODS verwenden, um die DCB-Unterparameter zur Verfügung zu stellen.

```
//STEP1    EXEC PGM=ERZEUGE
//GEN      DD  DSN=STEUER.BESCHEID(+1),DISP=(,CATLG),
//             UNIT=2314,SPACE=(80,200),
//             DCB=(PSEUDODS,BLKSIZE=400)
```

Der DCB-Parameter verweist auf die Datei, welche die DCB-Unterparameter enthält. Weitere DCB-Unterparameter können hinzugefügt bzw. überschrieben werden.

14.6 Dateien über mehrere Datenträger

Dateien, die so groß sind, daß sie auf einer Bandrolle oder einem Plattenstapel keinen Platz haben, können sich über mehrere Datenträger erstrecken. Mit dem Parameter VOL kann man ganz bestimmte Datenträger oder eine bestimmte Maximalzahl beliebiger Datenträger anfordern. Außerdem kann eine bestimmte Datenträgerfolgenummer angegeben werden, mit der die Verarbeitung beginnen soll. Mit Hilfe des Parameters UNIT können die Datenträger parallel geladen werden. Man kann mit UNIT die Lademeldung aber auch verzögern, bis die Datenträger tatsächlich gebraucht werden. Man sollte das parallele Laden nur benutzen, wenn es nicht zu viele Datenträger sind, und wenn sie sich schnell verarbeiten lassen. Im Multiprogramming-Betrieb sollte man das verzögerte Laden durchaus anwenden, um die E/A-Geräte auszulasten, besonders wenn die Verarbeitung langsam ist. Wenn die Anzahl der Datenträger größer als die Zahl der E/A-Geräte ist, muß das Laden sogar verzögert werden.

Verlängert man eine katalogisierte Datei über mehrere Datenträger, so muß die Disposition DISP=(MOD,CATLG) sein, selbst wenn die Datei schon katalogisiert war. CATLG bewirkt, daß die neuen Datenträgernummern beim Weiterschreiben der Datei dann auch im Katalog eingetragen werden.

Mit dem SPACE-Parameter muß für eine Datei auf einem Speicher mit direktem Zugriff Platz verlangt werden, wenn sie über weitere Datenträger ausgedehnt werden soll. Die Folgespureinrichtung kann in diesem Fall nicht benutzt werden. Um Plattendateien über weitere Datenträger auszudehnen, muß die Disposition MOD heißen.

Wenn der Platz auf dem Plattenstapel während des Schreibens aufgebraucht ist, erfolgt der Plattenwechsel. Bei Bändern erfolgt Rollenwechsel, wenn die Reflektormarke erreicht wird. Beim Lesen wird der Plattenstapel gewechselt, wenn ein Extent gelesen werden soll, der auf einem anderen Datenträger liegt. Bei Bändern erfolgt der Wechsel beim Lesen der EOV-Sätze (End of Volume). Das folgende Beispiel zeigt die Erzeugung, die Erweiterung und das Wiederauffinden von Dateien über mehrere Bänder bzw. mehrere Plattenstapel.

```
//STEP1    EXEC PGM=ERZEUGE
//A        DD   DSN=GROSS,DISP=(,CATLG),UNIT=BAND,
//             VOL=(PRIVATE,,,4,SER=(BD0100,BD0200))
```

Auf den Bändern BD0100 und BD0200 wird eine Banddatei erzeugt. Der Operator muß, falls erforderlich, 2 weitere Bänder bereitstellen und auf Anforderung laden. Alle benutzten Datenträger werden im Katalog registriert.

```
//B        DD  DSN=PLGROSS,DISP=(,CATLG),UNIT=PLATTE,
//             SPACE=(1600,(200,100))
```

Eine Plattendatei wird auf einem beliebigen Plattenstapel eingerichtet.

```
//STEP2    EXEC PGM=VERLAENG
//C        DD  DSN=GROSS,DISP=(MOD,CATLG),
//             UNIT=(,P,DEFER),VOL=PRIVATE
```

Alle Datenträger der Datei GROSS werden parallel geladen. Sollten weitere Datenträger erforderlich werden, muß sie der Operator zur Verfügung stellen und laden. Die Datenträgernummer jedes neuen Datenträgers wird in den Katalog eingetragen.

```
//D        DD  DSN=PLGROSS,DISP=(MOD,CATLG),UNIT=(,,DEFER)
```

Die Datei PLGROSS wird erweitert. Es werden so viele Datenträger verwendet, wie erforderlich sind.

```
//STEP3    EXEC PGM=AUFFIND
//E        DD  DSN=GROSS,DISP=OLD,UNIT=(,,DEFER),VOL=(,,2)
```

Die Datenträger werden, wie erforderlich, einer nach dem anderen geladen. Die Verarbeitung beginnt mit dem zweiten Datenträger.

```
//F        DD  DSN=PLGROSS,DISP=OLD,UNIT=(,,DEFER)
```

Die Datenträger werden, wie erforderlich, einer nach dem anderen geladen.

14.7 Dateischutz: RETPD, EXPDT, PASSWORD

Dateien können durch Angabe eines Aufbewahrungszeitraumes oder eines Kennwortes geschützt werden. Bevor Dateien, für die ein Aufbewahrungszeitraum angegeben wurde, verändert werden können, muß der Operator eine Antwort in das System eingeben. Sie können nicht ge-

löscht werden, es sei denn, man verwendet das in diesem Kapitel erläuterte Dienstprogramm IEHPROGM. Bevor eine mit Kennwort geschützte Datei eröffnet werden kann, muß der Operator das Kennwort angeben.

14.7.1 Schutz durch Aufbewahrungszeitraum: RETPD, EXPDT

Mit dem Parameter LABEL der DD-Karte kann Band- oder Plattendateien ein Aufbewahrungszeitraum zugeordnet werden. Bevor eine Datei, deren Aufbewahrungszeitraum noch nicht abgelaufen ist, verändert werden kann, muß der Operator seine Zustimmung geben. Wenn der Aufbewahrungszeitraum abgelaufen ist, unterscheidet sich die Datei von einer ohne Aufbewahrungszeitraum nicht mehr. Sie kann verändert werden oder gelöscht werden, ohne daß die Zustimmung des Operators erforderlich ist.

Die Systemdateien sollten mit einem Aufbewahrungszeitraum versehen werden, damit sie nicht versehentlich verändert oder gelöscht werden. Auf der anderen Seite sollte die Angabe eines Aufbewahrungszeitraumes sparsam verwendet werden, weil der Operator sonst mit der Eingabe seiner Zustimmung sehr beschäftigt werden kann.

Man gibt einen Aufbewahrungszeitraum an, indem man LABEL=RETPD=*tage* kodiert. Der Wert tage gibt die Anzahl der Tage an, die der Aufbewahrungszeitraum umfassen soll (maximal sind 9999 Tage zulässig). Es ist aber auch möglich, daß LABEL=EXPDT=yyddd kodiert wird. Der Wert yyddd ist das Datum, bis zu dem die Datei geschützt werden soll. Der Wert yy gibt das Jahr an und ddd den Tag. Wird kein Aufbewahrungszeitraum angegeben, so wird ein solcher von null Tagen angenommen.

14.7.2 Dateischutz durch Kennwort: PASSWORD

Man kann den Dateien ein gewisses Maß an Sicherheit geben, indem man sie mit einem Kennwort schützt. Kodiert man bei der Erzeugung der Datei den Unterparameter PASSWORD, so bedeutet das, daß zum Eröffnen der Datei ein Kennwort nötig ist. Der Dateiname und ein 8 Zeichen langes Kennwort müssen in der Systemdatei PASSWORD registriert werden. Man wird der Datei PASSWORD diesen Kennwortschutz ebenfalls geben. Bevor die Datei zum Lesen oder Schreiben eröffnet werden kann,

muß der Operator das Kennwort eingeben. Gelingt ihm das nach zwei Versuchen nicht, so wird der Job abnormal beendet.

Der Kennwortschutz bedeutet kein hohes Maß an Sicherheit. Ein geschickter Programmierer kann diesen Schutz umgehen, und zum andern, der Operator muß das Kennwort kennen. Der Kennwortschutz mag für einige Daten ausreichen, er ist aber für besonders wichtige Dateien, wie z.B. für Gehaltsdateien, nicht geeignet. Er ist für Informationen, die den Sicherheitsbestimmungen der Regierung unterliegen, ebenfalls ungeeignet. Will man den Kennwortschutz für eine Datei verlangen, so muß LABEL=(,,PASSWORD) kodiert werden.

Beide Schutzeinrichtungen, der Aufbewahrungszeitraum und das Kennwort, werden der Datei gewöhnlich bei ihrer Erzeugung zugeordnet. Der Aufbewahrungszeitraum läßt sich in jedem beliebigen Jobstep, der die Datei eröffnet, verändern oder auch löschen.

PASSWORD ist ein Positionsparameter, während RETPD und EXPDT Schlüsselwortparameter sind. Sie können in jeder beliebigen Zusammenstellung mit anderen LABEL-Unterparametern kodiert werden.

```
//A        DD  LABEL=(2,SL,PASSWORD,IN,RETPD=30),...
//A        DD  LABEL=(,,PASSWORD,EXPDT=99360),...
```

14.8 Steigerung der E/A-Leistungsfähigkeit

Zur Steigerung der E/A-Leistungsfähigkeit kann eine Reihe von DCB-Unterparametern benutzt werden. Im allgemeinen läuft ihre Anwendung darauf hinaus, daß man zwischen Speicherkapazität und Verarbeitungsgeschwindigkeit abwägen muß. Die Parameter SEP und AFF dienen dazu, die Dateien auf verschiedene Kanäle zu legen.

14.8.1 Schreibprüfung

Der DCB-Unterparameter OPTCD=W fordert für jede Plattendatei eine Schreibprüfung. Nach dem Schreiben werden die Daten unmittelbar wieder gelesen, um sicherzustellen, daß sie aus dem Speicher richtig übertragen wurden. Diese Prüfung verlangt eine zusätzliche Plattenumdrehung und ist natürlich sehr langsam. Bei einer 2321-Einheit wird diese

Prüfung stets durchgeführt, egal ob sie verlangt wurde oder nicht. Für andere Einheiten ist sie aber in der Regel nicht erforderlich.

14.8.2. Folgespureinrichtung

Die Einrichtung Folgespur dient zum Sparen von Plattenspeicherplatz. Normalerweise schreibt das System vollständige Blöcke auf jede Spur. Es überprüft dabei, ob der nächste Block noch vollständig auf dem Rest der Spur Platz hat. Falls das nicht der Fall ist, wird der nächste Block auf die nächste Spur geschrieben. Ist der Block größer als eine Spur, so beginnt jeder Block auf einer neuen Spur und erstreckt sich über so viele Spuren als nötig sind.

Wenn die Folgespureinrichtung zur Verfügung steht (bei 2314 und 2301 Standard, bei 2302, 2303, 2311 und 2321 wahlfrei), kann man mit dem DCB-Unterparameter RECFM=FT,RECFM=FBT, etc. verlangen, daß Blöcke, die nicht vollständig auf eine Spur passen, auf der nächsten Spur fortgesetzt werden. Dadurch wird Speicherplatz gespart; bei langen Sätzen sogar besonders viel.

Beispielsweise enthält eine 2311-Spur 3625 Bytes. Falls der Block 2000 Bytes lang ist, paßt nur ein Block auf eine Spur. Es werden 1625 Bytes verschwendet. Kodiert man aber DCB=RECFM=FBT, um mit der Folgespureinrichtung zu arbeiten, so wird dieser sonst verschwendete Platz ausgenutzt.

Im SPACE-Parameter sollte man den Unterparameter CONTIG benutzen, um die Bereiche zusammenhängend zuzuordnen. Da alle Spuren zusammenhängend sein sollen, kann sich eine solche Datei nicht über weitere Datenträger erstrecken. Für index-sequentielle Dateien, für FORTRAN-Dateien, die das Rückwärtssetzen benutzen, und für REGIONAL(3)-Dateien in PL/1 mit U oder V als Satzformat, kann die Folgespureinrichtung nicht verwendet werden. Auch bei Verwendung der DCB-Unterparameter BFTEK=E (Exchange Buffering) oder OPTCD=C (Chained Scheduling) kann die Folgespureinrichtung nicht verwendet werden.

14.8.3 Sätze vom Typ Spanned Records

Auf Speichern mit direktem Zugriff können Sätze variabler Länge die Blöcke überspannen (deshalb Spanned Records). Das bedeutet, daß ein Satz in Segmenten geschrieben wird, falls er nicht in einen Block hin-

einpaßt. Ist beispielsweise BLKSIZE=800 und sind zwei 300-Byte lange Sätze geschrieben worden, so wird ein dritter Satz, dessen Länge 600 Bytes betragen mag, aufgeteilt. Das erste Teilstück wird 200 Bytes betragen, um den angefangenen Block aufzufüllen, die restlichen 400 Bytes kommen in den nächsten Block. Folgt jetzt ein 1300 Byte Satz, so kommen die ersten 400 Bytes in den gerade angefangenen Block. Die folgenden 800 Bytes belegen den nächsten Block, während die restlichen 100 Bytes in den darauffolgenden Block gestellt werden.

Soll mit Sätzen vom Typ Spanned Records gearbeitet werden, so muß im DCB-Parameter RECFM=VS bzw. RECFM=VBS kodiert werden. Dieses Satzformat erlaubt eine wirksame Blockung, die vor allen Dingen von der Satzlänge unabhängig ist.

14.8.4 Standard-Blöcke

Eine Datei auf einem Speicher mit direktem Zugriff hat Standard-Blökke, wenn alle Blöcke (den letzten möglicherweise ausgenommen) die gleiche Länge haben, und wenn jeder Block vollständig auf einer Spur ist. Für den Fall der Standard-Blöcke darf weder die Folgespureinrichtung angefordert werden, noch darf die Blockgröße die Spurlänge überschreiten.

Der DCB-Unterparameter RECFM=FS oder RECFM=FBS sagt dem System, daß Standard-Blöcke vorliegen. Da alle Blöcke die gleiche Länge haben, und weil jeder Block vollständig auf seiner Spur ist, enthalten alle Spuren die gleiche Anzahl von Blöcken, die bezogen auf die Spur die gleiche relative Lage haben. Das System kann deshalb die genaue Plattenspeicheradresse eines jeden Blockes errechnen. Das spart beim Lesen oder Schreiben eine Umdrehung, und beschleunigt die Ein-Ausgabe deshalb beträchtlich.

14.8.5 Verkettete Kanalprogramme (Chained Scheduling)

Verkettet man mehrere E/A-Operationen miteinander, so kann man sie durch eine einzige E/A-Anforderung aufsetzen. Es ist beispielsweise möglich, mit einer einzigen E/A-Anforderung drei Blöcke in drei getrennte Puffer einzulesen. Die Blöcke werden nacheinander gelesen, ohne daß zwischen dem Einleseende des ersten Blockes und dem Einlesestart des zweiten Blockes eine Verzögerung liegt.

Um verkettete Kanalprogramme anzufordern, muß der DCB-Unterparameter OPTCD=C kodiert werden. Die Einrichtung verkettete Kanalprogramme erfordert etwa 2000 Bytes zusätzlichen Kernspeicher. Um sie wirksam einzusetzen, sollten wenigstens drei Puffer mit dem DCB-Unterparameter BUFNO zur Verfügung gestellt werden.

Die Einrichtung verkettete Kanalprogramme verringert die CPU-Zeit, die Kanal-Start/Stop-Zeit und die Rotationsverzögerung bei Plattenspeichereinheiten. Man sollte sie aber nur bei wirklich E/A-intensiven Jobs einsetzen. Da diese Einrichtung den Kanal sozusagen monopolisiert, sollten die anderen Dateien auf andere Kanäle gelegt werden. Bei Anforderung der Folgespureinrichtung oder einer anderen als der einfachen Pufferung und bei Benutzung von index-sequentiellen oder direkten Dateien sowie bei Benutzung des Lochstreifenlesers kann die Einrichtung verkettete Kanalprogramme nicht verwendet werden.

14.8.6 Kanaltrennung: SEP, AFF

Der *Kanal* ist der Weg, über den die Daten zwischen dem E/A-Gerät und dem Kernspeicher transportiert werden. Liest oder schreibt ein Jobstep mehrere Dateien über den gleichen Kanal, so kann dieser Andrang die Laufzeit des Jobs sehr verlängern. Mit den Parametern SEP und AFF ist es möglich, die Dateien über verschiedene Kanäle zu dirigieren. Natürlich sollte man sich vor der Benutzung von SEP und AFF über die Kanalkonfiguration der betreffenden Rechenanlage informieren.

Um eine Datei nicht über den von anderen Dateien des gleichen Steps benutzten Kanal zu schicken, kodiert man SEP=*ddname* oder SEP=(*ddname*,...,*ddname*). Die Angabe ddname ist der Name von bis zu acht DD-Karten. Sie müssen im gleichen Step vor der DD-Karte liegen, die SEP benutzt.

```
//STEP1    EXEC PGM=ALPHA
//A        DD   UNIT=2314,...
//B        DD   UNIT=2314,SEP=A,...
```

Die in der DD-Karte B beschriebene Datei geht nicht über den von der DD-Karte A benutzten Kanal.

```
//C        DD   UNIT=2314,SEP=(A,B),...
```

Die DD-Karte C belegt den von A und B benutzten Kanal nicht. Der Parameter SEP kann auf einer Folgekarte fortgesetzt werden, wenn die Unterbrechung nach einem vollständigen ddname, das Komma eingeschlossen, erfolgt.

```
//D        DD  UNIT=2314,SEP=(A,B),...
```

Die DD-Karte D besitzt die gleiche Kanaltrennung wie die Karte C.

Der Parameter AFF dient dazu, den Parameter SEP einer vorangegangenen Karte zu kopieren. Man kodiert AFF=*ddname*. Die DD-Karte D im obigen Step könnte unter Verwendung von AFF wie folgt kodiert werden:

```
//D        DD  UNIT=2314,AFF=C,...
```

Die Karte D muß nicht notwendigerweise den gleichen Kanal wie C benutzen. AFF bedeutet nur, daß der gleiche SEP-Parameter verwendet wird.

Das System ignoriert die Forderung nach Kanaltrennung, wenn dazu keine Möglichkeit besteht (beispielsweise zu wenig Kanäle). Das gleiche geschieht, falls eine Einheit mit ihrer Hardware-Adresse verlangt wird, falls ein Datenträger bereits geladen ist und durch AVR (Automatic Volume Recognition) lokalisiert wird, oder falls eine bestehende Datei auf einem permanently/resident Datenträger liegt.

Die Kanaltrennung sollte mit Bedacht eingesetzt werden. In einem Multiprogramming-System, in dem mehrere Jobs konkurrierend laufen, ist es durchaus so, daß die Auswirkung der Kanaltrennung für einen einzelnen Job nicht ins Gewicht fällt. Die Kanaltrennung beschränkt das System in der Freiheit Einheiten zuzuordnen und kann durchaus das unnötige Entladen von Datenträgern zur Folge haben. Man sollte die Kanaltrennung nur bei neuen Dateien bzw. bei alten Dateien auf ladbaren Datenträgern einsetzen, wenn man dabei bedeutende Einsparungen erwartet.

14.9 Direkte Dateien

Die direkte Organisationsform einer Datei gestattet es, jeden Satz unmittelbar zu adressieren, ohne daß seine Lage zu den anderen Sätzen der Datei von Bedeutung wäre. Jeder Satz enthält einen Ordnungsbegriff,

der seine Lage in der Datei bestimmt. Die Adressierung kann direkt oder indirekt erfolgen.

Bei der *direkten Adressierung* hat der Ordnungsbegriff eine direkte Beziehung zur absoluten bzw. relativen Spuradresse. Die *relative Spuradresse* einer Datei zählt die Spuren innerhalb der Datei, relativ zu ihrer ersten Spur. Läßt sich die Beziehung zwischen Ordnungsbegriff und Plattenplatz auch auf die *relative Satznummer* (Stellung des Satzes in der Spur) ausdehnen, so läßt sich die vollständige Blockadresse errechnen, wenn man nur die Adresse der ersten Spur und die Anzahl der Sätze pro Spur kennt.

Die zweite Methode der Adressierung, die *indirekte Adressierung* einer direkt organisierten Datei ist verwickelter. Um die Spuradresse aus dem Ordnungsbegriff abzuleiten, muß ein mathematisches Umrechnungsverfahren eingesetzt werden. Es bleibt dem Anwender überlassen, wie er die Sätze organisiert, und wie er diese Umformung von Ordnungsbegriff in Satzadresse bewerkstelligt.

Direkte Dateien können in den folgenden Sprachen erzeugt werden: COBOL (ORGANIZATION RELATIVE oder DIRECT, ACCESS RANDOM), FORTRAN (DEFINE FILE), PL/1 (REGIONAL DIRECT) und Assembler.

Die DD-Karten direkter Dateien unterscheiden sich von denen sequentieller Dateien wie folgt: Der Parameter SPLIT darf bei der Platzzuordnung nicht verwendet werden, es sind andere DCB-Unterparameter erforderlich. Im DCB-Parameter muß DSORG=DA oder DSORG=DAU kodiert werden, um dem System mitzuteilen, daß es sich um eine direkte Datei handelt. Werden die Sätze mit einem Schlüssel geschrieben, so muß auch KEYLEN im DCB-Parameter kodiert werden, um die Länge des Schlüssels anzugeben. Die Schlüssel haben eine feste Länge; hat die Datei Schlüssel, so hat jeder Satz einen solchen.

Direkte Dateien werden normalerweise wahlfrei verarbeitet. Die Sätze werden gelesen, gelöscht, hinzugefügt oder ersetzt. Besonders wichtig ist, daß der verarbeitete Satz völlig unabhängig von der Lage des gerade vorher verarbeiteten Satzes ist. Direkte Dateien können natürlich auch sequentiell nach aufsteigend geordneten Ordnungsbegriffen verarbeitet werden, wenn man bei bekannten Ordnungsbegriffen die Sätze nach ihren Ordnungsbegriffen aufsteigend sortiert. Der einzelne Satz kann in einer direkten Datei schneller lokalisiert werden als in einer index-sequentiellen Datei, weil kein Durchsuchen eines Index erforderlich ist. Auf der anderen Seite ist die direkte Organisationsform bei sequentieller Verarbeitung langsamer, weil die Queued-Zugriffstechnik

nicht verwendet werden kann. Jeder Satz muß einzeln lokalisiert werden, und die Übertragung der Sätze kann mit ihrer Verarbeitung nicht überlappt werden.

14.10 Index-sequentielle Dateien

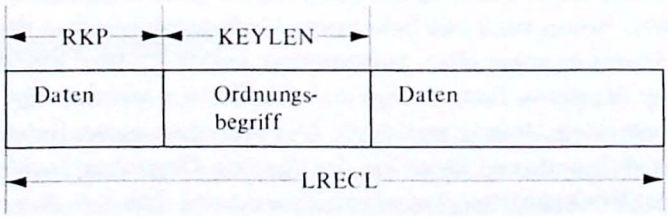
Index-sequentielle Dateien können im Assembler, in COBOL, PL/1 und RPG erzeugt werden. Zu der index-sequentiellen Datei kann direkt oder sequentiell zugegriffen werden. Jeder Satz muß einen ihn identifizierenden Ordnungsbegriff im Schlüsselteil des Satzes enthalten. Die Sätze sind nach den Ordnungsbegriffen sortiert gespeichert.

Beispielsweise könnte eine Bibliothek eine Datei besitzen, in der für jedes Buch ein Satz enthalten ist. Nehmen wir an, daß 150 000 Bücher vorhanden sind. Jeder Satz könnte den Titel des Buches, seinen Autor, den Verlag und andere Informationen enthalten. Der Titel könnte dann als Ordnungsbegriff benutzt werden, um die Bücher alphabetisch anzuordnen.

Wenn alle Sätze verarbeitet werden müssen, wie z.B. bei einer Inventur, ist die sequentielle Verarbeitung das Gegebene. Was aber, wenn nur ein Buch, etwa das mit dem Titel "Zelda's Zilch" in seinem Satz eine Veränderung verlangt. Wenn ein einzelner Satz verändert werden muß, ist der direkte Zugriff viel schneller. Genauso werden Bücher im direkten Zugriff zur Datei hinzugefügt oder gelöscht, ohne daß die ganze Datei verarbeitet werden muß.

14.10.1 Sätze

Index-sequentielle Dateien können nur auf Speichern mit direktem Zugriff eingerichtet werden. Die Datei besteht aus Sätzen, Blöcken, Spuren und Zylindern. Der Satz hat das folgende Format.

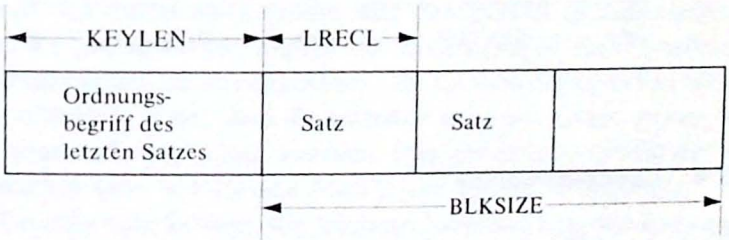


-RKP (Relative Key Position, relative Lage des Ordnungsbegriffes im Satz),
-KEYLEN (Länge des Ordnungsbegriffes; maximal 255 Bytes) und
-LRECL (Satzlänge) sind DCB-Unterparameter, deren Werte in Bytes angegeben werden.

Jeder Satz kann als aktiv (muß verarbeitet werden) oder inaktiv (belegt nur Platz oder zeigt einen gelöschten Satz an) gekennzeichnet werden. In unserem Beispiel aus der Bibliothek möge der Satz aus 40 Zeichen für den Titel, 20 Zeichen für den Autor, 20 Zeichen für den Verlag und noch einmal 20 Zeichen für andere Informationen bestehen. (RKP=0, KEYLEN=40, LRECL=100, läßt man RKP weg, so wird RKP=0 angenommen.) Die Sätze werden in der Reihenfolge der Ordnungsbegriffe (der Buchtitel) auf die Spuren gespeichert.

14.10.2 Blöcke

Um Plattenspeicherplatz zu sparen und um die Verarbeitungsgeschwindigkeit zu erhöhen, können mehrere Sätze zu einem Block zusammengefaßt werden. Der Ordnungsbegriff des letzten Satzes im Block wird dem gesamten Block vorangestellt, und im Schlüsselteil des Blockes gespeichert. Dadurch wird das Wiederauffinden der Ordnungsbegriffe erleichtert. Der Block hat folgendes Format.

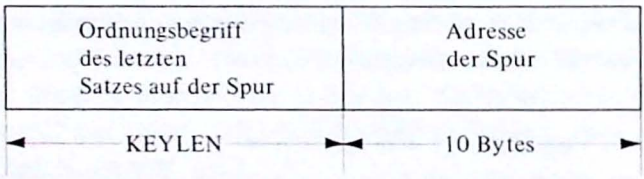


14.10.3 Spuren

Auf eine Spur kommen so viel ganze Blöcke wie es der Platz erlaubt. Die Gesamtzahl der Spuren, die solche Blöcke mit ihren Sätzen enthält, heißt *Hauptdatenbereich (Prime)*. Die Spuren liegen in Zylindern; die Zylinder müssen nicht notwendigerweise zusammenhängend sein. Für jeden Zylinder wird automatisch ein Spurindex auf der ersten Spur des Zylinders eingerichtet. Für jede Spur sind zwei Einträge vorhan-

den. Der erste Eintrag verweist auf die Spur, der zweite Eintrag, der sogenannte Überlaufeintrag, verweist auf den Überlaufbereich, für den Fall, daß Sätze aus der Spur rausgefallen sind.

Wird ein Satz zu der Datei hinzugefügt, so kommt er mit seinem Ordnungsbegriff auf eine bestimmte Spur. Alle Sätze dieser Spur, deren Ordnungsbegriffe größer sind als sein Ordnungsbegriff, werden verschoben, um ihm Platz zu machen. Fällt als Folge dieser Verschiebung ein Satz von der Spur, so wird er in den Überlaufbereich gestellt. Im Überlaufeintrag des Spurindex wird ein entsprechender Eintrag vorgenommen. Fallen mehrere Sätze von der Spur, so werden sie im Überlaufbereich logisch miteinander verbunden. Die Sätze im Überlaufbereich sind ungeblockt, und besitzen ein zusätzliches Feld von 10 Bytes. Dieses Feld wird für die logische Verbindung der Sätze untereinander benötigt. Wird ein Satz gelöscht, so bekommt er die Kennzeichnung inaktiv. Auf diese Art wird das zeitaufwendige Zurückschreiben des Satzes eingespart. Der Satz fällt erst aus der Datei raus, wenn er bei einer Verschiebung von der Spur fällt. Im Spurindex können ganze Spuren als inaktiv gekennzeichnet werden, um auf diese Weise zusätzlichen Platz zur Verfügung zu stellen. Der Eintrag im Spurindex hat das folgende Aussehen.



14.10.4 Überlaufbereiche

Es können zwei Überlaufbereiche einzeln oder zusammen angelegt werden, der Zylinderüberlaufbereich und der unabhängige Überlaufbereich. Der *Zylinderüberlaufbereich* besteht aus einer bestimmten Anzahl von Spuren auf jedem Zylinder, die für die Überlaufsätze bestimmt sind. Alle Sätze, die von den Spuren eines Zylinders herunterfallen, werden zuerst in den Zylinderüberlaufbereich gestellt. Da die Überlaufsätze im gleichen Zylinder bleiben, wird die Bewegung des Zugriffsmechanismus eingespart. Ist der Überlaufbereich eines Zylinders voll, so kann der unbenutzte Platz in anderen Zylindern nicht benutzt werden, weil die Zylinderüberlaufbereiche nicht teilbar eingerichtet sind.

Der *unabhängige Überlaufbereich* dagegen kann von allen Zylindern benutzt werden. Auf diese Weise wird Platz gespart. Für den unabhängigen Überlaufbereich wird eine eigene Platzzuordnung durchgeführt; sie heißt *Überlauf* (*Overflow*). Der Preis, der für diese Einrichtung bezahlt wird, heißt zusätzliche Zugriffszeit, denn der unabhängige Überlaufbereich liegt auf anderen Zylindern und ist vom Hauptdatenbereich getrennt.

Die beste Methode, Platz für Überlaufsätze bereitzustellen, besteht darin, dem Zylinderüberlaufbereich eine vernünftige Größe zu geben und einen unabhängigen Überlaufbereich anzulegen. Wenn die Zylinderüberlaufbereiche voll werden, wird dieser die Sätze aufnehmen können.

14.10.5 Zylinderindex

Erstreckt sich die Datei über mehr als einen Zylinder, so wird automatisch ein *Zylinderindex* eingerichtet. Er enthält für jeden Zylinder einen Eintrag, der das gleiche Format wie ein Eintrag im Spurindex hat.

14.10.6 Hauptindex

Es ist möglich einen *Hauptindex* einzurichten, wenn der Zylinderindex eine bestimmte Anzahl von Spuren überschreitet. Für jede Spur des Zylinderindex erfolgt dann im Hauptindex ein Eintrag. Der Hauptindex erübrigt es bei der Suche nach einem Satz den ganzen Zylinderindex zu durchsuchen. Er gestattet es, sofort die Eintragungen einer bestimmten Spur des Zylinderindex zu durchsuchen. Er ist besonders bei großen Dateien sehr nützlich. Über dem Hauptindex kann noch eine zweite und dritte Hauptindexstufe errichtet werden. Das geschieht, sobald die niedrigere Indexstufe eine bestimmte Anzahl von Spuren übersteigt. Assembler-Programme können die höchste Indexstufe in den Hauptspeicher bringen. Dadurch läßt sich die Suchzeit stark herabsetzen. Der für den Haupt- und Zylinderindex zugeordnete Bereich heißt *Indexbereich* (*Index*). Das Format einer Hauptindexeintragung ist das gleiche wie das einer Spurindexeintragung.

14.10.7 Zugriff zu index-sequentiellen Dateien

Die folgenden Schritte zeigen, wie das System zu einem Satz mit Hilfe seines Ordnungsbegriffes direkt zugreift.

1) Existiert kein Hauptindex, so beginnt die Suche mit Schritt 2. Existiert ein Hauptindex, so wird er nach dem ersten Eintrag durchsucht, der einen Ordnungsbegriff enthält, der größer als der gesuchte ist. Verweist der Hauptindex auf eine tiefere Indexstufe, so wird diese in ähnlicher Weise durchsucht. Der so aufgefundene Hauptindexeintrag verweist auf eine bestimmte Spur des Zylinderindex, die den gesuchten Ordnungsbegriff enthalten kann.

2) Existiert kein Zylinderindex, so besteht die Datei nur aus einem Zylinder, und die Suche geht mit Schritt 3 weiter. Existiert ein Zylinderindex, so wird die Spur, auf die Schritt 1 verwiesen hat, nach dem ersten Eintrag durchsucht, dessen Ordnungsbegriff grösser als der gesuchte ist. Der so aufgefundene Eintrag verweist auf einen Zylinder, der den gesuchten Ordnungsbegriff enthalten kann.

3) Der Spurindex, der die ersten Spuren des Zylinders belegt, wird nach dem ersten Eintrag durchsucht, dessen Ordnungsbegriff grösser als der gesuchte Ordnungsbegriff ist. Der auf diese Weise gefundene Spurindexeintrag verweist auf die Spur, die den gesuchten Satz enthalten kann.

4) Endlich wird die Spur nach dem Satz mit dem gesuchten Ordnungsbegriff durchsucht.

Um die Suche zu beschleunigen, sollten der Haupt- und Zylinderindex auf den gleichen Zylinder gelegt werden. Die Suchzeit läßt sich zusätzlich verringern, wenn man den Indexbereich (also Haupt- und Zylinderindex) auf einen anderen Datenträger als den Hauptdatenbereich legt. Wenn der Indexbereich aber auf den gleichen Datenträger gelegt werden muß, sollte er im Anschluß an den Hauptdatenbereich liegen oder noch besser mitten im Hauptdatenbereich. (Der Hauptdatenbereich wäre im letzten Fall nicht zusammenhängend.)

Eine index-sequentielle Datei wird sequentiell verarbeitet, indem jeder Zylinder über den Zylinderindex aufgesucht wird, und indem all seine Spuren sequentiell verarbeitet werden. Die sequentielle Verarbeitungszeit wird durch Sätze im Überlaufbereich stark erhöht, besonders wenn sie im unabhängigen Überlaufbereich liegen.

14.10.8 Erzeugung index-sequentieller Dateien

Eine index-sequentielle Datei kann eine Platzzuordnung für drei verschiedene Bereiche erforderlich machen: Für den Hauptdatenbereich,

den Indexbereich und den unabhängigen Überlaufbereich. Der Hauptdatenbereich muß stets vorhanden sein; der Indexbereich und der unabhängige Überlaufbereich sind wahlfrei. Die DD-Karten für eine index-sequentielle Datei müssen in der folgenden Reihenfolge kodiert werden:

```
//DDNAME    DD   DSN=NAME(INDEX),...
//          DD   DSN=NAME(PRIME),...
//          DD   DSN=NAME(OVFLOW),...
```

Die Angabe ddname wird nur auf der ersten DD-Karte kodiert. INDEX, PRIME und OVFLOW müssen stets in der gezeigten Reihenfolge kodiert werden, auch wenn INDEX oder OVFLOW fehlen. Eine temporäre Datei wird mit DSN=&&name(...) auf jeder der drei DD-Karten kodiert. Besteht die Datei nur aus dem Hauptdatenbereich, so braucht PRIME nicht kodiert zu werden. Das bedeutet, daß DSN=name und DSN=name (PRIME) gleichwertig sind, wenn die Datei mit nur einer DD-Karte erzeugt wird.

Der DCB-Unterparameter DSORG=IS oder DSORG=ISU muß auf jeder DD-Karte kodiert werden. Wird irgendein anderer DCB-Unterparameter auf einer Karte kodiert, so muß er auch auf den anderen beiden Karten kodiert werden. Das verzögerte Laden des Datenträgers ist nicht gestattet, das bedeutet, daß UNIT=(..,..,DEFER) nicht kodiert werden darf.

14.10.8.1 *Der Parameter SPACE*

Für index-sequentielle Dateien kann der Platz nach zwei Arten angefordert werden. Die einmal gewählte Methode muß für alle DD-Karten beibehalten werden. Um Platz anzufordern, kodiert man:

SPACE=(CYL,(*menge*,,*index*),,CONTIG)

Der Platz wird in Zylindern zugeordnet. Die Angabe index gibt die Anzahl der Zylinder für den Indexbereich an, wenn dieser in der Mitte des Hauptdatenbereiches liegen soll. CONTIG bewirkt, daß der Platz zusammenhängend ist. Diese Angabe muß auf allen Karten gemacht werden, wenn sie auf einer Karte gemacht worden ist. Wird mehr als eine Einheit angefordert, so wird der angegebene Platz auf jedem der Datenträger zugeordnet. Um ganz bestimmte Spuren anzufordern, kodiert man:

SPACE=(ABSTR,(*menge*,*adresse*,*index*))

Der Platz wird von einer ganz bestimmten Spur ab zugeordnet. Die Angabe menge muß eine ganze Anzahl von Zylindern umfassen. Die Angabe adresse muß sich auf die erste Spur eines Zylinders beziehen, dieser darf nicht der erste Zylinder des Datenträgers sein. Wurden mehr als eine Einheit verlangt, so wird der Platz angefangen von der verlangten Adresse zugeordnet. Die Zuordnung erstreckt sich solange über Folgedatenträger bis die Zuordnung die verlangte Menge erreicht hat. Die VTOC (Datenträgerinhaltsverzeichnis) des zweiten und folgenden Datenträgers muß jeweils im ersten Zylinder des Datenträgers liegen. Die Angabe index gibt die Anzahl der Spuren für den Indexbereich an. Diese Methode, den Platz von einer bestimmten Spur an zu verlangen, wird für index-sequentielle Dateien oft benutzt, weil die Lage der Hauptdaten-, Index- und Überlaufbereiche die Zugriffszeiten stark beeinflußt.

Index-sequentielle Dateien können nur dann mit DISP=(,CATLG) katalogisiert werden, wenn die Erzeugung mit einer DD-Karte geschah. Index-sequentielle Dateien mit mehr als einer DD-Karte können mit dem früher beschriebenen Dienstprogramm IEHPROGM katalogisiert werden, vorausgesetzt, alle DD-Karten haben den gleichen Einheitentyp verlangt.

14.10.8.2 *Beispiel einer Platzzuordnung*

Da die Platzzuordnung für index-sequentielle Dateien recht schwierig ist, soll das vorher angeführte Bibliotheksbeispiel dazu dienen, die einzelnen Schritte aufzuzeigen. Die Tabellen 3 und 4 von Kapitel 11 zeigen die für die Platzzuordnung benötigten Formeln:

$$\text{Blöcke/Spur} = 1 + \frac{\text{Spurkapazität} - \text{Bn}}{\text{Bi}}$$

Bn ist der letzte Block einer Spur, Bi ist jeder andere Block der Spur. Die Spurkapazität, Bi und Bn hängen vom Geräte-Typ ab. Für unser Beispiel wollen wir eine 2314 verwenden. Für diesen Fall lautet die Gleichung:

$$\text{Blöcke/Spur} = 1 + \frac{7294-(45+\text{KEYLEN}+\text{BLKSIZE})}{146+1.043(\text{KEYLEN}+\text{BLKSIZE})}$$

Die Bibliotheksdatei enthält 150 000 Sätze; jeder von ihnen ist 100 Bytes lang. Der Titel dient als Ordnungsbegriff und enthält 40 Bytes. Wenn wir annehmen, daß der Block 20 Sätze enthält, lauten die DCB-Unterparameter wie folgt: BLKSIZE=2000,LRECL=100,KEYLEN=40.

14.10.8.3 *Hauptdatenbereich*

Der Hauptdatenbereich kann auf mehr als einem Datenträger vom gleichen Typ liegen. Aller Platz, der in den letzten Zylindern des Hauptdatenbereiches nicht benutzt wird, wird als unabhängiger Überlaufbereich verwendet. Der Hauptdatenbereich wird in ganzen Zylindern zugeordnet. Die ersten Spuren eines Zylinders enthalten den Spurindex. (Der Datenträger, von dem das System geladen wird, kann nicht der erste für den Hauptdatenbereich zugeordnete Datenträger sein.)

1) Berechne die Anzahl der Sätze pro Spur des Hauptdatenbereiches.

$$\text{Blöcke/Spur} = 1 + \frac{7294-(45+\text{KEYLEN}+\text{BLKSIZE})}{146+1.043(\text{KEYLEN}+\text{BLKSIZE})}$$

$$= 1 + \frac{7294-(45+40+2000)}{146+1.043(40+2000)} = 3.29$$

Das bedeutet, auf eine Spur gehen 3 Blöcke.

Sätze/Spur = (Blöcke/Spur) · (Sätze/Block) = 3 · (20) = 60

2) Berechne die Spuren pro Zylinder, die für den Spurindex nötig sind. Diese Sätze sind ungeblockt und bestehen aus dem Ordnungsbegriff plus 10 Bytes Daten.

$$\text{Spurindexeintragungen/Spur} = 1 + \frac{7294-(45+\text{KEYLEN}+10)}{146+1.043(\text{KEYLEN}+10)}$$

$$= 1 + \frac{7294-(45+40+10)}{146+1.043(40+10)} = 37.33$$

Das bedeutet, eine Spur enthält 37 Indexeintragungen.

$$\text{Spuren/Zylinder für den Spurindex} = \frac{2(\text{Spuren/Zylinder})+1}{\text{Indexeintragungen/Spur}+2}$$

Tabelle 3 von Kapitel 11 zeigt, daß ein 2314-Plattenstapel 20 Spuren pro Zylinder enthält.
Damit beträgt der Spurindex folgende Spuren/Zylinder:

$$\frac{2(20)+1}{37+2} = \frac{41}{39}$$

Die ersten 1 2/39 Spuren jedes Zylinders werden für den Spurindex verwendet. Falls der noch unbenutzte Platz auf der zweiten Spur nämlich 37/39 Spuren noch Datenblöcke enthalten kann, muß

man ihn für die Bestimmung der Anzahl der Datensätze pro Zylinder in Rechnung stellen.

$$\text{Blöcke/Teilspur} = 1 + \frac{(37/39)\ (7294)-(45+40+2000)}{146+1.043(40+2000)} = 3.13$$

Das ergibt 3 Blöcke für den Rest von Spur 2 jedes Zylinders. Die ersten zwei Spuren eines jeden Zylinders enthalten also neben dem Spurindex noch 3 Blöcke oder 60 Sätze.

3) Man berechne den Platz, der je Zylinder für den Zylinderüberlaufbereich zugeordnet werden soll (falls ein solcher eingerichtet wird). Die Sätze im Überlaufbereich sind ungeblockt und haben ein zusätzliches 10 Byte langes Verbindungsfeld.
Überlaufsätze/Spur =

$$= 1 + \frac{7294-(45+\text{KEYLEN}+\text{LRECL}+10)}{146+1.043(\text{KEYLEN}+\text{LRECL}+10)}$$

$$= 1 + \frac{7294-(45+40+100+10)}{146+1.043(40+100+10)} = 24.47$$

Das bedeutet, auf eine Spur passen 24 Überlaufsätze.

Für wie viele Überlaufsätze pro Zylinder soll nun aber Platz vorgesehen werden? Die Größe dieses Platzes hängt davon ab, wie oft wir Sätze hinzufügen werden und wie kritisch die Faktoren Speicherplatz und Zugriffszeit sind. Man erreicht eine schnellere Zugriffszeit auf Kosten von Speicherplatz, wenn der Zylinderüberlaufbereich größer angelegt wird. Wir wollen unterstellen, daß 5% der Sätze hinzugefügt werden, und daß ferner die meisten von ihnen im Zylinderüberlaufbereich Platz haben sollen, damit die Zugriffszeit klein gehalten wird. Wenn wir die Anzahl der Sätze pro Zylinder grob abschätzen und davon 5% nehmen, so können wir die Zahl der Zylinderüberlaufsätze abschätzen und damit die Zahl der Spuren pro Zylinder bestimmen.

Überlaufsätze/Zylinder =

(Datenspuren/Zylinder) (Datensätze/Spur) · Prozentsatz

Nun können wir aber die Zahl der Datensätze pro Zylinder nicht errechnen, weil wir die Zahl der Überlaufsätze nicht kennen. Die Zahl der Datensätze hängt davon ab, wieviel Überlaufsätze zugelassen werden und damit wie groß der Überlaufbereich gewählt wird. Wir können die Zahl der Datensätze aber abschätzen. Ein 2314-Zy-

linder enthält 20 Spuren. Gut eine Spur wird für den Zylinderindex verwendet und mindestens eine Spur wird für den Zylinderüberlaufbereich verwendet werden. Das bedeutet, es bleiben 18 Spuren für die Datensätze übrig. (Wir werden die Rechnung wiederholen, wenn das Ergebnis zu weit von der 5% Grenze entfernt ist.)

Überlaufsätze/Zylinder = 18 · 60 · 0,5 = 54

Überlaufspuren/Zylinder =

$$\frac{\text{Überlaufsätze/Zylinder}}{\text{Überlaufsätze/Spur}} = \frac{54}{24} = 2.25$$

Das bedeutet, daß 2 Überlaufspuren pro Zylinder eingerichtet werden müssen (es ist nicht nötig genauer zu sein, denn 4,7% liegt nahe genug an 5%).

In der DD-Karte werden die DCB-Unterparameter OPTCD=Y und CYLOFL=*spuren* kodiert, um einen Zylinderüberlaufbereich anzufordern und um auf jedem Zylinder die betreffenden Spuren zuzuordnen. (CYLOFL=99 ist der maximale Wert.)

4) Man errechne nun die Gesamtzahl der für den Hauptdatenbereich benötigten Zylinder.

Sätze/Zylinder = (Sätze/Spur)(Datenspuren)+Datensätze im Spurindex

Datenspuren=Spuren/Zylinder-Spuren für Spurindex-Spuren für Überläufe

Sätze/Zylinder = 60 · (20-2-2)+60 = 1020

$$\text{Gesamtzahl der Zylinder} = \frac{\text{Anzahl der Sätze}}{\text{Sätze/Zylinder}} = \frac{150\,000}{1020} = 147.06$$

Das bedeutet, daß für die 150 000 Sätze 148 Zylinder benötigt werden.

14.10.8.4 *Indexbereich*

Falls der Hauptdatenbereich mehr als 1 Zylinder belegt, ist ein Zylinderindex nötig. Um die Suchzeit zu verringern, kann ein Hauptindex angelegt werden. Der Indexbereich umfaßt den Zylinder- und Hauptindex. Er kann in den Hauptdatenbereich oder an sein Ende gelegt werden. Der Indexbereich kann aber auch mit einer extra DD-Karte angelegt wer-

den. Wird der Platz gesondert angefordert, dann braucht der Indexbereich nicht auf dem gleichen Geräte-Typ liegen wie der Hauptdatenbereich. Er muß aber auf einem einzigen Datenträger untergebracht werden. Aller Platz, der in den letzten Zylindern des Indexbereiches nicht benötigt wird, wird als unabhängiger Überlaufbereich ausgenutzt. Voraussetzung dafür ist natürlich, daß der Indexbereich in diesem Fall auf dem gleichen Geräte-Typ wie der Hauptdatenbereich liegt.

5) Man errechne den für den Zylinderindex benötigten Platz (falls der Hauptdatenbereich also mehr als 1 Zylinder belegt). Für die Eintragungen im Zylinderindex wird die gleiche Formel benutzt wie zur Berechnung der Spurindexeintragungen. Da der gleiche Geräte-Typ verwendet wird, passen 37 Zylinderindexeintragungen auf eine Spur.

   Spuren für den Zylinderindex =

   $$\frac{\text{Anzahl der Zylinder} + 1}{\text{Eintragungen/Spur}} = \frac{148+1}{37} = 4.03$$

   Das bedeutet, daß der Zylinderindex 5 Spuren belegt.

6) Man errechne den für den Hauptindex aufzuwendenden Platz (falls gewünscht). Mit den DCB-Unterparametern OPTCD=M und NTM=*spuren* wird ein Hauptindex angefordert. NTM gibt die Anzahl der Zylinderindexspuren an, für die ein Eintrag im Hauptindex vorgenommen werden soll (NTM=99 ist der Maximalwert). Es muß jetzt entschieden werden, wie groß der Wert für NTM werden soll. Das bedingt wieder ein Abwägen zwischen Speicherplatz und Zugriffszeit. Kleinere NTM-Werte verringern die Zugriffszeit, erhöhen aber den Speicherbedarf. Vielleicht sind 2 Spuren Zylinderindex für je einen Hauptindexeintrag ein guter Kompromiß. Die Hauptindexeintragungen haben die gleiche Größe wie die Zylinderindexeintragungen; es werden also 37 Eintragungen auf eine Spur passen.

   Spuren für den Hauptindex der 1. Stufe =

   $$\frac{\text{Zylinderindexspuren/NTM+1}}{\text{Eintragungen/Spur}}$$

   (Das gilt, falls der Zylinderindex mehr als NTM Spuren umfaßt.)

   Spuren für den Hauptindex der 2. Stufe =

   $$\frac{\text{Hauptindexspuren der 1. Stufe/NTM} + 1}{\text{Eintragungen/Spur}}$$

(Das gilt, falls der Hauptindex der 1. Stufe mehr als NTM Spuren umfaßt.)

Spuren für den Hauptindex der 3. Stufe =

$$\frac{\text{Hauptindexspuren der 2. Stufe/NTM} + 1}{\text{Eintragungen/Spur}}$$

(Das gilt, falls der Hauptindex der 2. Stufe mehr als NTM Spuren umfaßt.)

$$\text{Spuren für den Hauptindex der 1. Stufe} = \frac{4.03/2+1}{37} = 0.08$$

Ein Hauptindex für die 2. und 3. Stufe wird nicht erzeugt. Die für den Indexbereich benötigten Spuren sind also 5 Spuren für den Zylinderindex und 1 Spur für den Hauptindex. Zusammen macht das 6 Spuren aus.

14.10.8.5 *Unabhängiger Überlaufbereich*

Der unabhängige Überlaufbereich nimmt die Sätze auf, die in den Zylinderüberlaufbereichen keinen Platz mehr haben. Der unabhängige Überlaufbereich kann mit einer eigenen DD-Karte angelegt werden. Das muß aber auf dem gleichen Geräte-Typ wie dem des Hauptdatenbereichs geschehen. Außerdem muß der unabhängige Überlaufbereich auf nur einem Datenträger liegen.

7) Man berechne die Größe des unabhängigen Überlaufbereichs (falls ein solcher angelegt wird). Die Größe des unabhängigen Überlaufbereichs hängt davon ab, wieviel Sätze hinzugefügt werden sollen, und wieviel Platz bereits im Zylinderüberlaufbereich reserviert worden ist. Für den Zylinderüberlaufbereich waren 5% festgelegt worden. Wir wollen für den unabhängigen Überlaufbereich deshalb ebenfalls 5% unterstellen.

Anzahl der Zylinder im unabhängigen Überlaufbereich =

Zylinder des Hauptdatenbereichs · 5% = 148 · 0.05 = 7.4

Diese Zahl wird auf 7 abgerundet, denn genauer zu rechnen ist nicht erforderlich.

Die Gesamtplatzanforderung ist jetzt die folgende:
- 148 Zylinder für den Hauptdatenbereich,
- 6 Spuren für den Indexbereich und
- 7 Zylinder für den unabhängigen Überlaufbereich.

Da ein 2314-Plattenstapel nur 200 Zylinder besitzt, sollte man vielleicht für den Hauptdatenbereich mehr als einen Plattenstapel nehmen.

14.10.9 Verwendung index-sequentieller Dateien

Um eine bestehende index-sequentielle Datei zu beschreiben, müssen DSN, DISP, UNIT und VOL kodiert werden. Ebenso muß der DCB-Unterparameter DSORG=IS bzw. DSORG=ISU kodiert werden, es sei denn, die Datei wurde weitergereicht (PASS). UNIT und VOL dürfen fehlen, wenn die Datei auf einem einzigen Datenträger liegt und katalogisiert oder übergeben (PASS) ist. Der Parameter DSN wird ohne die Angaben INDEX, PRIME oder OVFLOW kodiert. Liegt der Indexbereich auf einem eigenen Datenträger, so muß für ihn eine eigene DD-Karte kodiert werden.
Die folgenden Beispiele zeigen verschiedene Arten der Platzzuordnung für die index-sequentielle Datei. Sie zeigen gleichzeitig wie die DD-Karten kodiert werden müssen, damit die Datei wiederaufgefunden werden kann.

1) Getrennte Platzzuordnung für den Indexbereich, den Hauptdatenbereich und den unabhängigen Überlaufbereich.

```
//STEP1    EXEC PGM=ERZEUGEN
//A        DD   DSN=ISDAT(INDEX),DISP=(,KEEP),UNIT=2314,
//             VOL=SER=DA0200,SPACE=(CYL,1,,CONTIG),
//             DCB=(DSORG=IS,OPTCD=YM,CYLOFL=2,NTM=2)
```

Die Datei heißt ISDAT. Für den Indexbereich sind nur 6 Spuren erforderlich. Da aber der Platz in ganzen Zylindern zugeordnet werden muß, wird ein Zylinder verlangt. Kodiert man auf einer der drei DD-Karten CONTIG, dann muß auf jeder der anderen auch CONTIG kodiert werden.

```
//           DD   DSN=ISDAT(PRIME),DISP=(,KEEP),UNIT=(2314,2),
//             VOL=SER=(DA0300,DA0400),DCB=*.A,
//             SPACE=(CYL,74,,CONTIG)
```

Auf jedem der beiden Datenträger werden 74 Zylinder für den Hauptdatenbereich, zusammen also 148 Zylinder zugeordnet.

```
//          DD  DSN=ISDAT(OVFLOW),DISP=(,KEEP),UNIT=2314,
//              VOL=SER=DA0500,DCB=*.A,
//              SPACE=(CYL,7,,CONTIG)
```

Die DCB-Unterparameter müssen auf allen 3 DD-Karten übereinstimmen.

```
//STEP2     EXEC PGM=LESEN
//B         DD  DSN=ISDAT,DISP=OLD,UNIT=(2314,4),DCB=DSORG=IS,
//              VOL=SER=(DA0200,DA0300,DA0400,DA0500)
```

Die Datenträgernummern müssen in der gleichen Reihenfolge wie bei der Erzeugung der Datei angegeben werden.

2) Eigener Indexbereich und unabhängiger Überlaufbereich am Ende des Hauptdatenbereichs.

```
//STEP1     EXEC PGM=ERZEUGEN
//A         DD  DSN=ISDAT(INDEX),...
//          DD  DSN=ISDAT(PRIME),DISP=(,KEEP),UNIT=(2314,2),
//              VOL=SER=(DA0300,DA0400),DCB=*.A,
//              SPACE=(CYL,78,,CONTIG)
```

Der Platz für den Hauptdatenbereich wurde erhöht, um den unabhängigen Überlaufbereich mit unterbringen zu können.

```
//STEP2     EXEC PGM=LESEN
//B         DD  DSN=ISDAT,DISP=OLD,UNIT=(2314,3),DCB=DSORG=IS,
//              VOL=SER=(DA0200,DA0300,DA0400)
```

3) Indexbereich am Ende des Hauptdatenbereichs und unabhängiger Überlaufbereich.

```
//STEP1    EXEC PGM=ERZEUGEN
//A        DD  DSN=ISDAT(PRIME),DISP=(,KEEP),UNIT=(2314,2),
//             VOL=SER=(DA0300,DA0400),SPACE=(CYL,74,,CONTIG),
//             DCB=(DSORG=IS,OPTCD=YM,CYLOFL=2,NTM=2)
//         DD  DSN=ISDAT(OVFLOW),...
//STEP2    EXEC PGM=LESEN
//B        DD  DSN=ISDAT,DISP=OLD,UNIT=(2314,3),DCB=DSORG=IS,
//             VOL=SER=(DA0300,DA0400,DA0500)
```

4) Indexbereich im Hauptdatenbereich und eigener unabhängiger Überlaufbereich.

```
//STEP1    EXEC PGM=ERZEUGEN
//A        DD  DSN=ISDAT(PRIME),DISP=(,KEEP),UNIT=2314,
//             VOL=SER=DA0400,SPACE=(CYL,(149,,1),,CONTIG),
//             DCB=(DSORG=IS,OPTCD=YM,CYLOFL=2,NTM=2)
```

Der Indexbereich von einem Zylinder liegt in der Mitte des Hauptdatenbereichs. Da der Indexbereich auf einem Datenträger liegen muß, muß demzufolge auch der Hauptdatenbereich auf einem Datenträger liegen.

```
//         DD  DSN=ISDAT(OVFLOW),...
//STEP2    EXEC PGM=LESEN
//B        DD  DSN=ISDAT,DISP=OLD,UNIT=(2314,2),DCB=DSORG=IS,
//             VOL=SER=(DA0400,DA0500)
```

5) Indexbereich am Ende des Hauptdatenbereichs und unabhängiger Überlaufbereich am Ende des Indexbereichs.

```
//STEP1    EXEC PGM=ERZEUGEN
//A        DD  DSN=ISDAT,DISP=(,CATLG),UNIT=(2314,2),
//             VOL=SER=(DA0300,DA0400),SPACE=(CYL,78,,CONTIG),
//             DCB=(DSORG=IS,OPTCD=YM,CYLOFL=2,NTM=2)
```

Der Hauptdatenbereich wurde vergrößert, um Platz für den Indexbereich und den unabhängigen Überlaufbereich zu schaffen. Da die Datei mit einer einzigen DD-Karte erzeugt wurde, kann sie katalogisiert werden.

```
//STEP2    EXEC PGM=LESEN
//B        DD   DSN=ISDAT,DISP=OLD,DCB=DSORG=IS
```

6) Indexbereich in der Mitte des Hauptdatenbereichs und unabhängiger Überlaufbereich am Ende des Hauptdatenbereichs.

```
//STEP1    EXEC PGM=ERZEUGEN
//A        DD   DSN=&&ISDAT,DISP=(,PASS),UNIT=2314,
//             VOL=SER=DA0300,SPACE=(CYL,(156,,1),,CONTIG),
//             DCB=(DSORG=IS,OPTCD=YM,CYLOFL=2,NTM=2)
```

Der Hauptdatenbereich wurde vergrößert, um Platz für den eingebetteten Indexbereich und den am Ende liegenden unabhängigen Überlaufbereich bereit zu stellen. Die Datei muß auf einem einzigen Datenträger liegen. Index-sequentielle Dateien können temporär sein; sie können mit PASS übergeben werden, falls sie mit einer einzigen DD-Karte erzeugt wurden.

```
//STEP2    EXEC PGM=LESEN
//B        DD   DSN=*.STEP1.A,DISP=(OLD,DELETE)
```

Wenn die index-sequentielle Datei übergeben wurde, ist der Unterparameter DSORG=IS nicht nötig.

Zur Erzeugung einer index-sequentiellen Datei kann jede dieser sechs Methoden verwendet werden. Die auszuwählende Methode wird davon abhängen, wie groß der Hauptdatenbereich, der Indexbereich und der unabhängige Überlaufbereich sind. Sie wird außerdem davon abhängen, wo diese Bereiche liegen sollen.

14.10.9.1 *Indexbereich auf einem anderen Einheitentyp*

Man kann den Indexbereich auf eine Einheit legen, die einen anderen Typ als die Einheit des Hauptdatenbereichs hat. Beispielsweise könnten wir den Indexbereich auf einen 2311-Plattenstapel legen. Zu diesem

Zweck müssen wir den Platz neu errechnen. Die Formel von Tabelle 4 in Kapitel 11 lautet für den Platz auf einem 2311-Plattenstapel wie folgt:

$$\text{Blöcke/Spur} = 1 + \frac{3625-(20+\text{KEYLEN}+\text{LRECL})}{81+1.049(\text{KEYLEN}+\text{LRECL})}$$

$$\text{Indexeintragungen/Spur} = 1 + \frac{3625-(20+40+10)}{81+1.049(40+10)} = 27.64$$

Das bedeutet, auf eine Spur passen 27 Indexeintragungen.

$$\text{Spuren für den Zylinderindex} = \frac{148+1}{27} = 5.52$$

$$\text{Spuren für den Hauptindex} = \frac{5.52/2+1}{27} = 0.14$$

Ein Hauptindex der 2. und 3. Stufe ist nicht nötig. Der Gesamtplatz für den Indexbereich beträgt deshalb 7 Spuren (6+1). Ein 2311-Zylinder enthält 10 Spuren - ein Zylinder reicht deshalb aus. Als Beispiel für die Erzeugung und das Wiederauffinden einer index-sequentiellen Datei, die auf 2 verschiedenen Einheitentypen liegt, soll die Reorganisation einer index-sequentiellen Datei dienen. Wir wollen annehmen, daß die index-sequentielle Datei bereits besteht, daß ihr Überlaufbereich aber voll geworden ist.

14.10.9.2 *Reorganisation index-sequentieller Dateien*

Weil die Überlaufbereiche sich füllen und weil die Verarbeitung der Überlaufsätze länger dauert, müssen index-sequentielle Dateien periodisch reorganisiert werden. Das IBM-Dienstprogramm IEBISAM kopiert eine index-sequentielle Datei, löscht Sätze, die das Löschbyte 'FF' im ersten Datenbyte haben, und macht die Überlaufbereiche wieder frei, indem alle Sätze in den Hauptdatenbereich gebracht werden. IEBISAM wird mit der folgenden JCL ausgeführt:

```
//GO        EXEC PGM=IEBISAM,PARM=COPY,REGION=12K
```

IEBISAM steht in der SYS1.LINKLIB. PARM=COPY sagt dem Programm, daß die Datei kopiert werden soll (IEBISAM besitzt Anwendungen, die im Rahmen dieses Buches nicht besprochen werden). Der Parameter REGION ist nur in einem MVT-System erforderlich.

```
//SYSPRINT DD  SYSOUT=A
```

Zur Ausgabe des Protokolls ist eine Druckdatei erforderlich.

```
//SYSUT1    DD  DSN=ISDAT,DISP=(OLD,DELETE),UNIT=2311,
//              VOL=SER=DA0100,DCB=DSORG=IS
//          DD  DSN=ISDAT,DISP=(OLD,DELETE),UNIT=(2314,3),
//              VOL=SER=(DA0300,DA0400,DA0500),DCB=DSORG=IS
```

SYSUT1 beschreibt die zu reorganisierende Datei. Zur Beschreibung einer index-sequentiellen Datei, die auf Geräten zweier verschiedener Typen liegt, sind 2 DD-Karten erforderlich. Da die Datei kopiert wird, kann die alte Datei gelöscht werden.

```
//SYSUT2    DD  DSN=ISDAT(INDEX),DISP=(,KEEP),
//              UNIT=(2311,SEP=SYSUT1),
//              SPACE=(CYL,1),DCB=*.SYSUT1
//          DD  DSN=ISDAT(PRIME),DISP=(,KEEP),
//              UNIT=(2314,2,SEP=SYSUT1),
//              SPACE=(CYL,74),DCB=*.SYSUT1
//          DD  DSN=ISDAT(OVFLOW),DISP=(,KEEP),
//              UNIT=(2314,SEP=SYSUT1),
//              SPACE=(CYL,7),DCB=*.SYSUT1
```

SYSUT2 beschreibt die reorganisierte Datei. Sie muß auf anderen Datenträgern als die alte Datei liegen.

14.11 Zusammenfassung

Dieses Kapitel schließt die Diskussion der Job Control-Sprache ab. Wie der Umfang des Buches zeigt, kann die JCL ganz schön schwierig werden. Sie ist auf der anderen Seite jedoch außerordentlich nützlich.

Viele Details der JCL sind schwierig zu behalten, besonders die natürlich, die selten verwendet werden. Der Anhang C enthält deshalb zum schnellen Nachschlagen eine Kurzzusammenfassung jeder der JCL-Karten. Der Anhang gibt außerdem die Textseiten an, auf denen die betreffenden JCL-Parameter im Detail besprochen werden.

Anhang A: DCB-Unterparameter

BFALN — Beginn eines jeden Puffers auf Vollwort- (F wie Fullword) oder Doppelwortgrenze (D wie Doubleword).

BFTEK — Die vom Data Management zu verwendende Pufferungstechnik. (S wie Simple bedeutet einfache Pufferung, und E wie Exchange bedeutet Austauschpufferung; wird nur bei QSAM (Queued Sequential Access Method) verwendet.)

BLKSIZE — Maximale Länge eines Blockes in Bytes (eine Zahl). (Darf bei BISAM (Basic Indexed Sequential Access Method) nicht mehr angegeben werden, weil der Wert mit QISAM Laden festliegt.)

BUFL — Länge des Puffers in Bytes, der vom Puffer-Pool beschafft werden soll (eine Zahl).

BUFNO — Anzahl der Puffer, die dem Data Control Block zugeordnet werden sollen. (Sie kann bei ISAM (Indexed Sequential Access Method) nicht verwendet werden.)

BUFRQ — Anzahl der Puffer, die im voraus von der Warteschlange auf dem Speicher mit direktem Zugriff eingelesen werden sollen. (Wird nur bei den Zugriffsmethoden der Datenfernverarbeitung verwendet.)

CODE — Lochstreifen-Code, mit dem die Daten im Lochstreifen abgelocht werden.

- I - IBM BCD Lochstreifen- und Übertragungs-Code (8 Spuren).
- F - Friden-Code (7 Spuren).
- B - Burroughs-Code (7 Spuren).
- C - National Cash Register-Code (8 Spuren).

A - USASCII (8 Spuren).
T - Fernschreib-Code (5 Spuren).
N - Ohne Umsetzung.

CPRI — Die einer Sende- oder Empfangs-Operation zugeordnete relative Priorität. (Wird nur bei den Zugriffsmethoden der Datenfernverarbeitung verwendet.)

CYLOFL — Gibt die Anzahl der Spuren an, die auf jedem Zylinder für Überlaufsätze frei gehalten werden. (Nur bei QISAM (Queued Indexed-Sequential Access Method).)

DEN — Bandschreibdichte

0 - 200 Bits/Inch (nur 7-Spur-Bänder)
1 - 556 Bits/Inch (nur 7-Spur-Bänder)
2 - 800 Bits/Inch
3 - 1600 Bits/Inch (nur 9-Spur-Bänder)

Bei 7-Spur-Bändern müssen alle Informationen auf der Bandrolle (wie z.B. Kennsätze, Daten, Bandmarken) in der gleichen Dichte geschrieben werden. Für eine SYSOUT Datei darf DEN nicht angegeben werden.

DSORG — Dateiorganisation

PS - Physical Sequential - Sequentiell
PSU - Physical Sequential Unmovable - Sequentiell und unbeweglich
PO - Partitioned Organization - Organisationsform der untergliederten Datei
POU - Partitioned Organization Unmovable - Organisationsform der untergliederten Datei und unbeweglich
IS - Indexed-Sequential - Index-sequentiell
ISU - Indexed-Sequential Unmovable - Index-sequentiell und unbeweglich
DA - Direct Access - Organisationsform für direkten Zugriff
DAU - Direct Access Unmovable - Organisationsform für direkten Zugriff und unbeweglich

EROPT — Die im Fehlerfall auszuführende Option. Nur bei QSAM (Queued Sequential Access Method) anwendbar.

ACC - Annehmen (Accept)
SKP - Überspringen (Skip)
ABE - Abnormal beenden (Abnormal End of Task)

GDSORG — Organisation einer graphischen Datei. Wird nur bei den graphischen Zugriffsmethoden verwendet.

GNCP — Maximale Anzahl der E/A-Makro-Instruktionen, die vor einer WAIT-Makro-Instruktion gegeben werden können. Wird nur bei den graphischen Zugriffsmethoden verwendet.

HIARCHY — Die Nummer der Hierarchie-Stufe, in der der Puffer-Pool angelegt werden soll (0 oder 1).

INTVL — Die beabsichtigte Verzögerung in Sekunden zwischen den Durchläufen durch eine Polling-Liste. Nur bei den Zugriffsmethoden der Datenfernverarbeitung.

KEYLEN — Die Schlüssellänge in Bytes (eine Zahl). (Nicht erlaubt bei BISAM und QSAM (Basic Indexed-Sequential Access Method und Queued Sequential Access Method).)

LIMCT — Gibt die Suchgrenzen in Blöcken oder Spuren an, falls bei BDAM (Basic Direct Access Method) die Option "Ausgedehnte Suchoperation" (Extended Search) mit OPTCD=E verlangt wurde (eine Zahl).

LRECL — Tatsächliche oder maximale Länge eines logischen Satzes in Bytes (eine Zahl). Bei Sätzen vom Typ Variable-Spanned, bei denen die Länge des logischen Satzes 32756 überschreitet, muß LRECL=X angegeben werden. Kann bei BDAM und BISAM (Basic Direct Access Method und Basic Indexed-Sequential Access Method) nicht verwendet werden.

MODE — Gibt beim Kartenleser oder Kartenstanzer die Art der Verarbeitung an. (C Dualkarteneinrichtung - E gleich EBCDI-Code)

NCP — Die maximale Anzahl der vor einem CHECK-Makro ausgegebenen READ- und WRITE-Makros (Anzahl der Kanalprogramme). Kann bei den direkten oder sequentiellen Zugriffsmethoden nicht verwendet werden.

NTM — Gibt die Anzahl der Spuren des Zylinderindex an, für die im Hauptindex ein Eintrag vorgenommen werden soll. Die Angabe ist erforderlich, falls mit OPTCD=M ein Hauptindex verlangt worden ist. Ein Hauptindex verringert bei einem sehr großen Zylinderindex die Suchzeiten.

OPTCD — Zusätzlich vom Data Management zur Verfügung zu stellende Dienstleistungen.

- W - Verlangt eine Schreibprüfung.
- F - Im READ- oder WRITE-Makro des Programms wird Feedback (Rückmeldung) verlangt.
- E - Ausgedehnter Suchvorgang (Extended Search).
- R - Verwendung relativer Blockadressen bei BDAM (Basic Direct Access Method). Verwendung der Reorganisationskriterien in den DCB-Feldern RORG1, RORG2 und RORG3 bei QISAM (Queued Indexed-Sequential Access Method).
- T - Anforderung der Summenbildung.
- A - Verwendung der Plattenspeicheradressen. (MBBCCHHR).
- C - Verwende verkettete Kanalprogramme (Chained Scheduling).
- M - Richte einen Hauptindex ein.
- Y - Verwende einen Zylinderüberlaufbereich.
- I - Verwende einen unabhängigen Überlaufbereich.
- L - Lösche mit "FF" im ersten Datenbyte gekennzeichnete Sätze beim Zugang neuer Sätze zur Datei.
- U - Datenfehler ausschalten (Universal Character Set Universaltypensatz).
- Z - Wird ein Datenfehler für eine Magnetbandeingabedatei festgestellt, so wird ein verkürztes Fehlerbeseitigungsverfahren durchgeführt. Das gilt für die sequentiellen Zugriffsmethoden und für EXCP. Diese Einrichtung sollte nur dann verlangt werden, wenn man weiß, daß das Band Fehler enthält, und wenn es nicht notwendig ist, daß alle Sätze verarbeitet werden.

PRTSP — Zeilenvorschub für den Drucker (0, 1, 2 oder 3).

RECFM — Satzformat der Sätze der Datei. Die folgenden Abkürzungen können in allen erforderlichen Kombinationen verwendet werden.

- U - Sätze undefinierter Länge
- V - Sätze variabler Länge
- F - Sätze fester Länge
- B - Geblockte Sätze
- T - Folgespureinrichtung
- S - Standard-Blöcke bei Sätzen fester Länge. Das bedeutet, daß in der Datei weder verkürzte Blöcke noch nicht volle Spuren sind. Bei Sätzen variabler Länge bedeutet S, daß der logische Satz sich über mehr als einen Block erstrecken kann.
- A - ASA-Steuerzeichen
- M - Maschinensteuerzeichen

RKP — Gibt die relative Position des Ordnungsbegriffs im logischen Satz an (gezählt wird von 0 ab). Kann nur bei ISAM (Indexed-Sequential Access Method) angegeben werden.

SOWA — Größe eines vom Anwender zur Verfügung gestellten Arbeitsbereichs in Bytes. Gilt nur für Zugriffsmethoden der Datenfernverarbeitung.

STACK — Ablagefächer beim Kartenleser und Kartenstanzer (1 oder 2).

TRTCH — Aufzeichnungstechnik für 7-Spur-Magnetbänder. Die angegebenen Werte können in Kombination verwendet werden.

- C - Datenumsetzung
- E - Prüfung auf gerade
- T - Beim Lesen Übersetzung von BCD in EBCDIC; beim Schreiben Übersetzung von EBCDIC in BCD. Prüfung auf ungerade.
- ET - Beim Lesen Übersetzung von BCD in EBCDIC, beim Schreiben Übersetzung von EBCDIC in BCD. Prüfung auf gerade.

Abdruck aus der Publikation Job Control Language mit Genehmigung der International Business Machines Corporation (1969).

Anhang B: Steuerzeichen

Die Sätze einer Datei können jeweils im ersten Datenbyte ein Steuerzeichen enthalten, das den Drucker oder Stanzer steuert. (Bei Sätzen variabler Länge ist das erste Datenbyte das fünfte Byte jedes Satzes, das unmittelbar auf das 4 Byte große Satzbeschreibungsfeld folgt.) Das Steuerzeichen selbst wird weder angedruckt noch gestanzt.

Das Steuerzeichen kann entweder ein Maschinensteuerzeichen (RECFM=..M) oder ein ASA-Steuerzeichen (RECFM=..A) sein. Das Maschinensteuerzeichen ist je nach verwendetem Gerät verschieden, und hängt von diesem ab; man beziehe sich auf die betreffende IBM-Publikation der System Reference Library, die das Gerät beschreibt. Die ASA Steuerzeichen sind in der folgenden Tabelle aufgeführt:

EBCDIC Zeichen	Vorgang vor dem Schreiben des Satzes
b	Einzeiliger Vorschub vor dem Drucken (Blank Code)
0	Zweizeiliger Vorschub vor dem Drucken
-	Dreizeiliger Vorschub vor dem Drucken
+	Kein Vorschub vor dem Drucken
1	Vorschub zu Kanal 1
2	Vorschub zu Kanal 2
3	Vorschub zu Kanal 3
4	Vorschub zu Kanal 4
5	Vorschub zu Kanal 5
6	Vorschub zu Kanal 6
7	Vorschub zu Kanal 7
8	Vorschub zu Kanal 8
9	Vorschub zu Kanal 9
A	Vorschub zu Kanal 10
B	Vorschub zu Kanal 11
C	Vorschub zu Kanal 12
V	Auswahl Stanzablagefach 1
W	Auswahl Stanzablagefach 2

Diese Steuerzeichen schließen die von USASI FORTRAN definierten mit ein. Wird irgendein anderes Zeichen angegeben, so wird es als 'b' oder V verstanden, je nachdem welches Ausgabegerät verwendet wird. Es wird keine Fehleranzeige gesetzt.

Abdruck aus der Publikation Supervisor and Data Management Services mit Genehmigung der International Business Machines Corporation (1969).

Anhang C: Zusammenfassung der Job Control-Sprache

1 *JOB-Karte* (s.S. 53-62)

Die erste Karte in einem Job muß eine einzelne JOB-Karte sein.

// *jobname* JOB (*kostenstellennummer, abrechnungsinformationen*), *name, schlüsselwortparameter*

jobname (s.S. 54) - ein nicht verwechselbarer 1-8 Zeichen langer Name (A-Z, 0-9, @ $ #).

kostenstellennummer, abrechnungsinformationen (s.S. 55) - stellt für den Job Abrechnungsinformationen zur Verfügung, so wie sie von der betreffenden Installation vorgeschrieben werden. Ihre Angabe kann zwingend notwendig gemacht werden.

name (s.S. 56) - ein 1-20 stelliger Name zu Identifikationszwecken. Seine Angabe kann zwingend vorgeschrieben werden. Der Name muß in Hochkommas eingeschlossen werden, falls er die Zeichen Leerstelle, / ') (* + - = enthält.

MSGLEVEL=(*jcl, zuordnung*) (s.S. 57) - druckt JCL-Karten und Zuordnungsmeldungen an. Kodiert man für jcl 0, so wird nur die JOB-Karte angedruckt. Kodiert man 1, so werden alle JCL-Karten angedruckt. Im Fall von 2 werden nur die JCL-Karten des Eingabestroms angedruckt. Kodiert man für zuordnung 0, so werden die Zuordnungsmeldungen unterdrückt. Bei 1 werden alle Zuordnungsmeldungen angedruckt.

PRTY=*priorität* (s.S. 58) - nur MFT und MVT. Zahlenangabe für die Jobpriorität von 0-13; 13 ist der höchste Wert. Die Installation setzt einen unterstellten Wert fest.

MSGCLASS=*klassse* (s.S. 59) - nur MFT und MVT. Ausgabeklasse für Systemnachrichten. Der unterstellte Wert ist MSGCLASS=A.

CLASS=*klasse* (s.S. 60) - nur MFT und MVT. Die Jobklasse ist ein einzelner Buchstabe von A bis O. Die Klassen werden von der Installation vergeben. Der unterstellte Wert ist CLASS=A.

TYPRUN=HOLD (s.S. 60) - nur MFT und MVT. Diese Angabe hält einen Job in der Eingabe-Queue fest, bis er vom Operator freigegeben wird.

COND=((*zahl, operator*),...,(*zahl, operator*)) (s.S. 84) - gibt für die Ausführung nachfolgender Steps bis zu 8 Tests an, falls vorausgehende Steps abnormal enden. Der Wert für zahl kann zwischen 0 und 4095 liegen. Die Vergleichsoperatoren sind GT, GE, EQ, LT, LE und NE. Der Test wird am Ende eines jeden Steps durchgeführt. Ist irgendeine der Testbedingungen erfüllt, so werden alle noch verbleibenden Steps übersprungen und nicht ausgeführt. Die Kodierung kann unterbrochen und auf einer Folgekarte fortgesetzt werden, wenn die Unterbrechung nach einem vollständigen Test, das Komma eingeschlossen, erfolgt.

ROLL=(*auslagerbar, lagert aus*) (s.S. 84) - nur MVT. Man kodiert an den entsprechenden Stellen YES oder NO, um die Rollout/Rollin-Bedingungen anzugeben.

REGION=(*haupt* K,*hierarchie* K) (s.S. 86) - nur MVT. Die Werte haupt und hierarchie sind 1-5 stellige gerade Zahlen. Sie geben die 1024-Byte großen Blöcke des zuzuordnenden zusammenhängenden Speichers an. Ist die Anzahl ungerade, so rundet das System auf eine gerade Zahl auf.

RD= *wiederanlaufbedingung* (s.S. 89) - setzt die Bedingungen für den automatischen Wiederanlauf und steuert das Schreiben von Checkpoints: R (Wiederanlauf und Checkpoint schreiben), NC (weder Wiederanlauf noch Checkpoints schreiben), NR (kein Wiederanlauf, aber Schreiben von Checkpoints) und RNC (automatischer Wiederanlauf, aber kein Schreiben von Checkpoints).

RESTART=(*, *checkid*) (s.S. 91) - lasse den Job später wieder anlaufen. Der Wiederanlauf geschieht vom Checkpoint, der mit checkid genannt ist und oder vom in stepname genannten Step. Man kodiert *, um den Wiederanlauf mit dem ersten Step erfolgen zu lassen.

TIME=(*minuten, sekunden*) (s.S. 79) - nur MVT. Zeitbegrenzung für den Job. Die Angabe minuten kann zwischen 0 und 1440 liegen (1440 schaltet alle Zeitbegrenzung aus), die für sekunden zwischen 0 und 59. In der Installation werden gewisse Werte unterstellt.

2 *EXEC-Karte* (s.S. 63-78)

Eine EXEC-Karte nennt das auszuführende Programm oder ruft ein katalogisiertes Verfahren auf.

prozedur
PROC=*prozedur*
PGM=*programm*
PGM=*.*rückbezug*

//stepname EXEC ----------, *schlüsselwortparameter*

stepname (s.S. 64) - der 1-8 stellige Name, der den Step bezeichnet (A-Z, 0-9, @ $ #).

prozedur (s.S. 69) - der 1-8 stellige Name des katalogisierten Verfahrens (A-Z, 0-9, @, $,#).

programm (s.S. 64) - der 1-8 stellige Name des Programms (A-Z, 0-9, @, $,#).

*.rückbezug (s.S. 68) - der Rückbezug bezieht sich auf eine vorausgegangene DD-Karte, die das Programm beschreibt.
*.ddname
*.stepname.ddname
*.stepname.procstepname.ddname

PARM=*wert* (s.S. 71) - dem Jobstep werden Parameter übergeben. Unterparameter müssen durch Komma getrennt werden. Enthält der Parameter Unterparameter oder besondere Zeichen (Leerstelle, ./')(*&+-=), so muß er in Hochkommas eingeschlossen werden. Der Parameter darf nach einem abgeschlossenen Unterparameter, das Komma mitgezählt, abgebrochen werden, um auf einer Folgekarte weiterkodiert zu werden. In diesem Fall muß aber in Klammern kodiert werden. Unterparameter mit Sonderzeichen kommen zusätzlich in Hochkommas.

ACCT=(*abrechnungsinformationen*) (s.S. 74) - stellt dem Step Abrechnungsinformationen zur Verfügung, so wie sie von der Installation vorgeschrieben werden. Unterparameter werden durch Kommas

getrennt. Enthalten die Unterparameter Sonderzeichen (Leerstelle, ./')(*&+-=), so müssen sie in Hochkommas kodiert werden. Nach einem vollständigen Unterparameter, das Komma mitgezählt, darf unterbrochen werden, um auf einer Folgekarte weiter zu kodieren.

TIME=(*minuten, sekunden*) (s.S. 79) - nur MVT. Zeitbegrenzung für den Step. Die Angabe minuten kann zwischen 0 und 1440 liegen (1440 schaltet alle Zeitbegrenzung aus), die für sekunden zwischen 0 und 59. In der Installation werden gewisse Werte unterstellt.

DPRTY=(*v1, v2*) (s.S. 74) - nur MVT. Die Dispatching-Priorität wird aus 16 · (v1) + v2 errechnet. Die Zahlen v1 und v2 können zwischen 0 und 15 liegen.

EVEN
ONLY
zahl

COND=((----, *operator, stepname*), . . . , (*zahl, operator, stepname*)) (s.S. 82) - gibt bis zu 8 Tests an, um festzustellen, ob der Step übersprungen werden soll oder nicht. Bei EVEN wird der Step ausgeführt, auch wenn der vorausgegangene Step abnormal geendet hat. Bei ONLY wird der Step nur dann ausgeführt, wenn ein vorausgegangener Step abnormal geendet hat. Die Zahl kann zwischen 0 und 4095 liegen. Die Vergleichsoperatoren sind GT, GE, EQ, LT, LE und NE. Die Angabe stepname bewirkt, daß der Test nur mit dem genannten Step durchgeführt wird; fehlt diese Angabe, so wird der Test mit allen vorausgegangenen Steps durchgeführt. Ist irgendeiner der Tests erfüllt, so wird der Step übersprungen. Die Kodierung darf nach einem vollständigen Test, das folgende Komma mitgerechnet, unterbrochen werden, um auf einer Folgekarte fortgesetzt zu werden.

ROLL=(*auslagerbar, lagert aus*) (s.S. 84) - nur MVT. Man kodiert an den entsprechenden Stellen YES oder NO, um die Rollout/Rollin-Bedingungen anzugeben.

REGION=(*haupt* K, *hierarchie* K) (s.S. 86) - nur MVT. Die Werte haupt und hierarchie sind 1-5 stellige gerade Zahlen. Sie geben die 1024-Byte großen Blöcke des zuzuordnenden zusammenhängenden Speichers an. Ist die Anzahl ungerade, so rundet das System auf eine gerade Zahl auf.

RD=*wiederanlaufbedingung* (s.S. 89) - setzt die Bedingungen für den automatischen Wiederanlauf und steuert das Schreiben von Check-

points: R (Wiederanlauf und Checkpoint schreiben), NC (weder Wiederanlauf noch Checkpoint schreiben), NR (kein Wiederanlauf, aber Schreiben von Checkpoints) und RNC (automatischer Wiederanlauf, aber kein Schreiben von Checkpoints).

Auf EXEC-Karten, die ein katalogisiertes Verfahren aufrufen, können Schlüsselwortparameter kodiert werden, indem der Stepname aus der Prozedur an das Schlüsselwort angehängt wird. Die Parameter müssen in diesem Fall in der EXEC-Karte in der Reihenfolge kodiert werden, in der die Steps im Verfahren auftreten. Wird der Stepname nicht angehängt, so können die Parameter in beliebiger Reihenfolge kodiert werden. Sie gelten dann für alle Steps des Verfahrens. (s.S. 71)

3 *Besondere DD-Karten*

```
//JOBLIB   DD  DSN=NAME,DISP=SHR
```

(s.S. 66) - Unmittelbar hinter der JOB-Karte kann eine JOBLIB Karte liegen (Verknüpfung von Dateien ist möglich). Sie beschreibt eine Programmbibliothek, die die auszuführenden Programme enthält. Die Karte darf in einem katalogisierten Verfahren nicht liegen.

```
//STEPLIB  DD  DSN=NAME,DISP=SHR
```

(s.S. 67) - Irgendwo hinter die EXEC-Karte kann eine STEPLIB Karte gelegt werden (Verknüpfung von Dateien ist möglich). Sie beschreibt eine Programmbibliothek, die die auszuführenden Programme enthält. Diese Karte darf auch in einem katalogisierten Verfahren vorkommen.

```
//SYSUDUMP DD  SYSOUT=A
```

(s.S. 76) - Die SYSUDUMP Karte kann irgendwo hinter der EXEC-Karte eingefügt werden. Sie bewirkt einen Speicherabzug der Partition oder Region für den Fall, daß der Step abnormal endet.

```
//SYSABEND DD  SYSOUT=A
```

(s.S. 76) - Sie wirkt wie die SYSUDUMP Karte, liefert aber zusätzlich einen Speicherabzug des Nucleus.

```
//SYSCHK   DD  SCHLUESSELWORTPARAMETER
```

(s.S. 94) - Die SYSCHK Karte liegt unmittelbar hinter der JOB-Karte und der JOBLIB Karte. Sie beschreibt die Datei, welche die Checkpoints enthält.

4 *DD-Karte* (s.S. 96-135, 155-203)

Für jede im Step benutzte Datei muß nach der EXEC-Karte eine DD-Karte eingefügt werden.

```
                DATA
                DUMMY
//ddname    DD -------, schlüsselwortparameter
```

ddname (s.S. 99) - ein 1-8 stelliger Name, der im Step jede DD-Karte unverwechselbar identifiziert (A-Z, 0-9, @, $, #).

* (s.S. 155) - Der Stern zeigt an, daß im Eingabestrom Datenkarten folgen. Diese Karten dürfen in den Spalten 1 und 2 kein /* oder // enthalten.

DATA (s.S. 155) - Das Schlüsselwort zeigt an, daß im Eingabestrom Datenkarten folgen. Diese Karten dürfen in den Spalten 1 und 2 // enthalten; /* jedoch ist nicht erlaubt.

DUMMY (s.S. 167) - ordnet einer sequentiellen Datei den Status einer Pseudodatei zu. Um den Status einer Pseudodatei zuzuordnen, kann man auch DSN=NULLFILE kodieren.

DDNAME=*ddname* (s.S. 157) - verschiebt die Definition einer Datei bis eine DD-Karte mit dem angegebenen ddname angetroffen wird.

SYSOUT=(*klasse, programm, formnummer*) (s.S. 160) - leitet Dateien über den Ausgabestrom. Die Angabe Klasse kann A bis Z sein; SYSOUT=A ist gewohnheitsgemäß eine Druckausgabe und SYSOUT=B eine Stanzausgabe. Programm ist ein Anwenderprogramm zur Verarbeitung der Ausgabe. Die Formnummer ist eine 1-4 stellige Dezimalzahl zur Bezeichnung des betreffenden Endlosformulars.

UCS=(*code*,FOLD,VERIFY) (s.S. 165) - gibt den Zeichensatz für einen 1403 Drucker an, der die Einrichtung Universaltypensatz besitzt. Der Wert Code gibt den Zeichensatz wie folgt an: AN, HN, PCAN, PCHN, PN, QN, QNC, RN, SN, TN, XN oder YN. FOLD bewirkt das Umklappen des ersten, zweiten und dritten Quadranten des EBCD Interchange Codes in den vierten Quadranten. Die Angabe VERIFY bewirkt die Anzeige des Zeichensatzes, um eine Nachprüfung zu ermöglichen.

name
**.rückbezug*

DCB=(----------,*unterparameter*,...,*unterparameter*) (s.S. 100) - gibt Parameter an, die die Definition der Datei vervollständigen. Name ist der Name einer katalogisierten Datei, von der die DCB-Unterparameter kopiert werden sollen. Der Rückbezug bezieht sich auf eine DD-Karte, die die zu kopierenden DCB-Unterparameter enthält. Die Unterparameter können in jeder beliebigen Reihenfolge kodiert werden; sie sind im Anhang A beschrieben. Der DCB-Parameter kann nach einem vollständigen Unterparameter - das folgende Komma eingeschlossen - unterbrochen werden, um ihn auf einer Folgekarte fortzusetzen.

DSN=*name* (s.S. 105) - benennt die Datei. Name kann 1 - 8 Stellen lang sein (A-Z, 0-9, @, $, #, -, 12/0-Doppellochung). Zur Kennzeichnung einer temporären Datei werden dem Namen zwei Zeichen && vorangestellt.

DSN=*name*	DSN=&&*name*
DSN=**.rückbezug*	
DSN=*name*(*glied*)	DSN=&&*name*(*glied*)
DSN=*name*(*zahl*)	
DSN=*name*(INDEX)	DSN=&&*name*(INDEX)
DSN=*name*(PRIME)	DSN=&&*name*(PRIME)
DSN=*name*(OVFLOW)	DSN=&&*name*(OVFLOW)

typ
gruppe P
adresse *volumes*

UNIT=(------,-------,DEFER,SEP=(*ddname*,...,*ddname*)) (s.S. 109) - (s.S. 109) - der Parameter gibt das E/A-Gerät an.
typ (s.S. 110) - gibt den Typ des E/A-Gerätes an.
gruppe (s.S. 112) - ist der Name, den die Installation einer Gruppe von E/A-Geräten zugeordnet hat.
adresse (s.S. 110) - Hardware-Adresse des E/A-Gerätes.
P (s.S. 114) - lädt die Datenträger einer katalogisierten Datei parallel.
volumes (s.S. 114) - gibt die Anzahl der parallel zu ladenden Datenträger an.
DEFER (s.S. 114) - verzögert das Laden des Datenträgers bis zum Eröffnen der Datei.

SEP (s.S. 115) - die Datei wird nicht auf die Einheiten gelegt, die bereits den mit ddname benannten 1-8 DD-Karten zugeordnet sind. Der Parameter kann nach einem vollständigen ddname, das Komma eingeschlossen, unterbrochen werden, um auf einer Folgekarte fortgesetzt zu werden.

UNIT=AFF=*ddname* (s.S. 113) - ordnet die Datei einem E/A-Gerät zu, dem bereits die in der DD-Karte ddname beschriebene Datei zugeordnet ist.

DISP=(*status, normale disposition, abnormale disposition*) (s.S. 116) - gibt den Status der Datei zu Stepbeginn an und verfügt die Disposition der Datei.

	KEEP	
NEW	DELETE	KEEP
MOD	PASS	DELETE
OLD	CATLG	CATLG
SHR	UNCATLG	UNCATLG

DISP=(-----,-----------,----------)

NEW (s.S. 117) - erzeugt eine Datei.
MOD (s.S. 117) - erzeugt oder verlängert eine Datei.
OLD (s.S. 118) - liest bzw. erweitert eine bestehende Datei bei ausschließlicher Kontrolle
SHR (s.S. 118) - die gelesene Datei ist anderen Jobs zugänglich.
KEEP (s.S. 121) - eine nicht-temporäre Datei wird gehalten.
DELETE (s.S. 122) - löscht eine Datei.
PASS (s.S. 120) - übergibt die Datei den nachfolgenden Steps.
CATLG (s.S. 122) - die Datei wird gehalten und katalogisiert.
UNCATLG (s.S. 123) - die Datei wird entkatalogisiert, aber noch gehalten.

VOL=(PRIVATE,RETAIN, *folgenummer, anzahl*,

REF=*name*
REF=*.*rückbezug*
SER=(*datenträgernummer, . . . ,datenträgernummer*)
----------) (s.S. 126) -

gibt Informationen über den Datenträger.

PRIVATE (s.S. 130) - ein ladbarer Datenträger vom Typ Private.
RETAIN (s.S. 131) - hält Datenträger vom Typ Private zwischen den Steps.
folgenummer (s.S. 131) - gibt die Folgenummer der katalogisierten Datei an, mit der die Verarbeitung beginnen soll (1-4stellig).
anzahl (s.S. 132) - ist die Anzahl der Datenträger, über die sich die Datei ausdehnen kann (1-255).
REF (s.S. 128) - ordnet den von einer anderen Datei benutzten Datenträger zu.
SER (s.S. 128) - gibt die 1-6 stellige Datenträgernummer an (A-Z, 0-9, @, $, #, -). Die Datenträgernummer muß in Hochkommas eingeschlossen werden, falls sie die Sonderzeichen Leerstelle, . / ')(* & += enthält. Der Parameter kann nach einer Datenträgernummer, das Komma eingeschlossen, unterbrochen werden, um auf einer Folgekarte fortgesetzt zu werden.

```
          CYL
          TRK                    index
          blocklänge             directory
SPACE=(---------,(menge,zuschlag,--------),
                ALX
                MXIG
                CONTIG
       RLSE,-------, ROUND)
```

(s.S. 172) - fordert Platz auf einem Speicher mit direktem Zugriff an.

blocklänge (s.S. 173) - fordert den Platz in Blöcken an.
TRK (s.S. 175) - fordert den Platz in Spuren an.
CYL (s.S. 175) - fordert den Platz in Zylindern an.
menge (s.S. 172) - Anzahl der Einheiten der Primär-Zuordnung.
zuschlag (s.S. 176) - Anzahl der Einheiten der Sekundär-Zuordnung.
directory (s.S. 178) - Anzahl der 256-Byte Sätze, die für das Inhaltsverzeichnis der untergliederten Datei zugeordnet werden sollen. Man kann etwa 5 Namen oder Zweitnamen pro Block annehmen.
index (s.S. 178) - Anzahl der Zylinder für den Indexbereich, wenn dieser im Hauptdatenbereich einer index-sequentiellen Datei liegt.
RLSE (s.S. 179) - gibt überflüssigen Platz beim Schließen der Datei frei.

CONTIG (s.S. 180) - ordne zusammenhängenden Platz zu.

MXIG (s.S. 180) - ordne den größten freien Platz auf dem Datenträger zu, der gleich dem oder größer als der verlangte Platz ist.

ALX (s.S. 180) - ordne die 5 größten freien und zusammenhängenden Bereiche auf dem Datenträger zu, die gleich dem oder größer als der verlangte Platz sind.

ROUND (s.S. 181) - der verlangte Platz wird auf ganze Zylinder aufgerundet.

index
directory

SPACE=(ABSTR,(*menge, adresse*,--------)) (s.S. 182) - ordne die angegebenen Spuren zu. Der Wert menge ist die Anzahl der zuzuordnenden und zusammenhängenden Spuren. Der Wert adresse ist die Spuradresse der ersten Spur.

SPLIT=(*anzahl*,CYL,(*menge, zuschlag*)) (s.S. 182) - Zylinderaufteilung zwischen mehreren Dateien.

Der Wert anzahl ist die Zahl der Spuren auf jedem Zylinder, die der ersten Datei zugeordnet werden sollen. Der Wert menge gibt die Zahl der Zylinder an, die allen Dateien zusammen zugeordnet werden sollen. Um den anderen Dateien Spuren zuzuordnen, wird auf den folgenden DD-Karten SPLIT=*anzahl* kodiert.

SPLIT=(*prozent, blocklänge*,(*menge, zuschlag*)) (s.S. 183) - Zylinderaufteilung zwischen mehreren Dateien. Der Wert prozent ist der Prozentsatz der Spuren, die auf jedem Zylinder der ersten Datei zugeordnet werden sollen. Der Wert menge ist die Anzahl der Blöcke, die allen Dateien zusammen zugeordnet werden sollen. Um den anderen Dateien Spuren zuzuordnen, wird auf den folgenden DD-Karten SPLIT=*prozent* kodiert. Der Prozentsatz muß zwischen 1 und 100 liegen. Die Summe der einzelnen Prozentsätze darf 100 nicht überschreiten.

CYL
TRK
blocklänge

SUBALLOC=(-------- ,(*menge, zuschlag, directory*),*rückbezug*) (s.S. 184) - der Platz der durch den Rückbezug angegebenen Datei wird weitervergeben.

IN RETPD=*nnnn*
OUT EXPDT=*yyddd*

LABEL=(*dateifolgenummer, typ*, PASSWORD, ----, ----------)
(s.S. 193 und 242) - gibt Informationen über den Kennsatz und den Dateischutz an.

dateifolgenummer (s.S. 193) - relative Stellung der Datei auf dem Magnetband.
typ (s.S. 194) - Kennsatztyp: SL, NL, NSL, SUL, BLP.
PASSWORD (s.S. 242) - Dateischutz durch ein Kennwort.
IN (s.S. 198) - eröffnet eine sequentielle Datei nur für Eingabe.
OUT (s.S. 198) - eröffnet eine sequentielle Datei nur für Ausgabe.
RETPD (s.S. 243) - die Datei erhält einen Aufbewahrungszeitraum in Tagen (0-9999).

EXPDT (s.S. 243) - es wird das Datum, an dem der Aufbewahrungszeitraum abläuft, angegeben.

SEP=(*ddname, . . . , ddname*) (s.S. 247) - die Datei wird nicht über die von anderen Dateien benutzte Kanäle geleitet. Die Werte ddname beschreiben 1-8 dieser anderen Dateien. Der Parameter kann nach einem ddname, das Komma mitgerechnet, unterbrochen werden, um auf einer Folgekarte fortgesetzt zu werden.

AFF=*ddname* (s.S. 247) - der Parameter SEP einer vorausgegangenen DD-Karte wird kopiert.

5 *Delimiter-Karte*

/* (s.S. 155) - sie wird hinter die Eingabedatenkarten im Eingabestrom gelegt, um das Ende der Datei anzuzeigen.

6 *PROC-Karte*

//name PROC *schlüsselwortparameter*

(s.S. 147) - die erste Karte in einem katalogisierten Verfahren kann eine PROC-Karte sein. Sie unterstellt den symbolischen Parametern Standardwerte. Die symbolischen Parameter enthalten 1-7 Zeichen (A-Z, 0-9); das erste muß A-Z sein. Der symbolische Parameter im Verfahren beginnt mit dem Zeichen &. Die den symbolischen Parametern zugeordneten Werte können beliebig lang

sein; sie können die Zeichen A-Z, 0-9, @, $,# und Leerstelle, ./ ')(*+-= enthalten. (Hochkommas nötig, falls die Zeichen ab Leerstelle verwendet werden.)

7 *Befehls-Karte*

// befehl operanden (s.S. 228) - Befehls-Karten dürfen vor einer JOB-, EXEC-, Null- oder anderen Befehls-Karte liegen. Sie dienen dazu, Operatorbefehle mit dem Eingabestrom einzugeben.

8 *Null-Karte*

// (s.S. 229) - Eine Null-Karte kann an das Ende des Jobs gelegt werden, um anzuzeigen, daß der Job zu Ende ist. Legt man 4 Null-Karten hinter den letzten Job, so wird dieser eingelesen, ohne daß der Reader abschließt.

9 *Kommentar-Karte*

//* (s.S. 229) - Kommentar-Karten können hinter die JOB-Karte irgendwo zwischen die JCL-Anweisungen gelegt werden.

Literaturhinweis

Das Material dieses Buches basiert auf den folgenden IBM-Publikationen der System Reference Library.

1. A22-6810, IBM System/360 System Summary
2. C20-1649, Student Text Introduction to IBM System/360 Direct Access Storage Devices and Organization Methods.
3. C20-1684, Student Text Introduction to IBM Data Processing Systems.
4. C24-3337, Report Program Generator Language.
5. C26-3756, Assembler (F) Programmer's Guide.
6. C27-6909, Graphic Programming Services for IBM 2250 Display Unit.
7. C27-6912, Graphic Programming Services for IBM 2260 Display Station (Local Attachment).
8. C27-6926, Multiprogramming with a Fixed Number of Tasks (MFT), Concepts and Considerations.
9. C27-6935, Planning for Rollout/Rollin.
10. C27-6939, Planning for Multiprogramming with a Fixed Number of Tasks (MFT).
11. C27-6942, Introduction to Main Storage Hierarchy Support for IBM 2361 Models 1 and 2.
12. C28-6380, COBOL(F) Programmer's Guide.
13. C28-6534, IBM System/360 Operating System Introduction.
14. C28-6535, Concepts and Facilities.
15. C28-6538, Linkage Editor.
16. C28-6539, Job Control Language.
17. C28-6540, Operator's Guide.
18. C28-6550, System Programmer's Guide.
19. C28-6554, System Generation.
20. C28-6586, Utilities.

21. C28-6594, PL/I (F) Programmer's Guide.
22. C28-6628, System Control Blocks
23. C28-6632, Job Control Language Charts.
24. C28-6646, Supervisor and Data Management Services.
25. C28-6647, Supervisor and Data Management Macro Instructions.
26. C28-6656, Checkpoint Restart.
27. C28-6680, Tape Labels.
28. C28-6702, Planning Guide for the Loader.
29. C28-6708, Advanced Checkpoint/Restart Planning Guide.
30. C28-6817, FORTRAN IV (G and H) Programmer's Guide.
31. C33-4000, ALGOL Programmer's Guide.
32. Y28-6607, System/360 Operating System Direct-Access Device Space Management. Program Logic Manual, Program Number 360S-DS-508, Restricted Distribution.

Sachwortverzeichnis

Abnormale Beendigung, Dumps 76
Abnormale Disposition, Parameter DISP 124
Abrechnungsinformationen
  EXEC-Karte 74
  JOB-Karte 55
ABSTR, SPACE-Unterparameter zur Anforderung bestimmter Spuren 182
AFF-Parameter der DD-Karte für die Kanaltrennung 247
AFF, UNIT-Unterparameter 113
ALIAS, Steuerkarte des Linkage Editor 209
Allocation, Platzzuordnung für Speicher mit direktem Zugriff, Primary, Secondary 172
alphanumerische Zeichen 32
ALX, SPACE Unterparameter 180
Andrucken der JCL Karten 57
  der Kommentar-Karten 229
  der überschreibenden Karten 139
Anforderung von zusammenhängendem Platz 180
Anwendungsprogramme 21
Ausführung 28
Ausführungs-Step 44
Ausgabeklassen 160
Ausgabestrom 24

Bandmarke 192
Bandschreibdichte 36, 191, 199
Basic-Zugriffstechnik 35
Befehls-Anweisung 228
Bibliotheken 27
Bit 31
BLDA, Steuerkarte für IEHPROGM 235
BLDG, Steuerkarte für IEHPROGM 236
BLDX, Steuerkarte für IEHPROGM 234
BLKSIZE, DCB-Unterparameter für die Blocklänge 101
  index-sequentielle Dateien 251
  Dateien im Eingabestrom 157
  Dateien im Ausgabestrom 161
  SPACE-Parameter 173
  konventionelle E/A-Geräte 164
Block 33
Blocken 34
blocklänge, blockweise Anforderung von Platz 173
Blockungsfaktor 34
BLP, LABEL-Unterparameter für Überspringen der Kennsatzverarbeitung 196
BUFNO, DCB-Unterparameter; gibt Anzahl der Puffer an 102
Byte 31

CATLG, IEHPROGM Steuerkarte 233
CATLG, DISP-Unterparameter zum Katalogisieren von Dateien 122
Chained Scheduling 246
Checkpoint 89
Checkpoint Identifikation 89
CHKPT-Makro 89
CLASS, Parameter der JOB-Karte 60
CODE, DCB-Unterparameter 165
Compilation 27
Compiler 27
Compile-Step Beispiel 40
COND-Parameter für das Überspringen von Jobsteps 81
CONNECT, IEHPROGM Steuerkarte 235
CONTIG, SPACE-Unterparameter zur Anforderung von zusammenhängenden Platz 179, 180
Control Volume 230, 235
CYL, SPACE-Unterparameter zur Anforderung ganzer Zylinder 175
CYLOFL, DCB-Unterparameter für den Zylinderüberlaufbereich 259

DASDI, Direct-Access Device Initialization 171
DATA, Parameter der DD-Karte für Daten im Eingabestrom 155
Data Control Block 97
Data Management 20
Datei 30
Datei über mehrere Datenträger 117, 123, 241
Dateien im Eingabestrom 155
Dateien im Ausgabestrom 160
Dateifolgenummer 193
Dateiname 105
Datenbestand (File) 30
Datenfernverarbeitung bei Rollout/Rollin 85
Datenspeicherung 35
Datenträger 38
Datenträgerkennsatz 127
Datenträgerwechsel (Plattenwechsel) 241
DCB-Parameter 100
  BUFNO-Unterparameter 102, 247
  CODE-Unterparameter 165
  CYLOFL-Unterparameter 259
  Dateien im Eingabestrom 155
  Dateien im Ausgabestrom 161
  DEN-Unterparameter 199
  DSORG-Unterparameter 101, 249, 255
  Generation Data Groups 238
  Konventionelle E/A-Geräte 164
  Kopieren von Parametern 103
  LRECL-Unterparameter 101, 251
  MODE-Unterparameter 165
  NTM-Unterparameter 260
  OPTCD-Unterparameter 244, 247, 259
  PRTSP-Unterparameter 164
  RECFM-Unterparameter 102, 165, 245 246
  RKP-Unterparameter 251
  Rückbezug 103
  STACK-Unterparameter 165
  SYSOUT 161
  TRTCH-Unterparameter 200
DD-Karte 96
  Format 98
  Hinzufügung zu katalogisierten Verfahren 138
  Löschen von Parametern in katalogisierten Verfahren 141
  Modifikation von Parametern in katalogisierten Verfahren 138
  Plazierung 137, 153
  Überschreibung von Parametern in katalogisierten Verfahren 138
ddname (Data Definition Name) 41, 99
  Kodierung bei den höheren Programmiersprachen 99
DDNAME, nachträgliche Definition von Dateien 157
  verknüpfte Dateien 159
DEFER, UNIT-Unterparameter für verzögertes Laden 114
DELETE, DISP-Unterparameter zum Löschen von Dateien 122
DEN, DCB-Unterparameter zur Angabe der Bandschreibdichte 199
Dienstprogramme 21
direkte Adressierung für direkte Dateien 249
direkte Dateien 248
direkte Dateiorganisation 31, 248
DISP-Parameter 116
  abnormale Disposition 124
  normale Disposition 119
  Status der Datei 116
Dispatching-Priorität 58, 74
Disposition
  der Banddatei 120
  der Plattendatei 120
  nicht durchgeführt 120
DLTA, Steuerkarte für IEHPROGM 235
DLTX, Steuerkarte für IEHPROGM 234
DPRTY, Parameter der EXEC-Karte 74
DSN, Parameter der DD-Karte für den Dateinamen 105
  Generation Data Groups 108
  index-sequentielle Dateien 108
  qualifizierte Namen 107
  Sonderzeichen im Namen 107
DSORG, DCB-Unterparameter für die Dateiorganisation 101
Dualkarteneinrichtung 165
DUMMY, Parameter der DD-Karte für Pseudodateien 167
  mit dem DDNAME-Parameter 159
  mit dem UNIT=AFF-Parameter 113
  mit dem UNIT=SEP-Parameter 116
  mit dem VOL=SER-Parameter 129
Dumps bei abnormaler Beendigung 76
dynamische Struktur 28

EBCDIC Zeichen 32
Ein-/Ausgabe 30
Eingabestrom 24

Eingabestrom-Dateien 155
ENTRY, Steuerkarte des Linkage Editor 209
Erzeugung index-sequentieller Dateien 254
Erzeugung katalogisierter Verfahren 173
EVEN, COND-Unterparameter 83
EXEC-Karte 63
  Format 63
  Zusammenfassung 77, 94
EXPDT, LABEL-Unterparameter 243
Extent der Platzzuordnung 176
externe Referenzen 211, 206, 27

FOLD, UCS-Unterparameter zum Umklappen der Code-Quadranten 166
Folgespureinrichtung 175, 245
FORTRAN Compile; Link und Go-Verfahren 207
Fortsetzung der Steuerkarte der Dienstprogramme 232
Fortsetzung von JCL-Karten 51
  ACCT-Parameter 74
  COND-Parameter 82
  DCB-Parameter 102
  Dienstprogramm Steuerkarten 232
  Linkage Editor Steuerkarten 209
  PARM-Parameter 72
  SEP-Parameter 247
  SEP-Unterparameter 115
  symbolische Parameter 148
  VOL-Parameter 128

Generation Data Groups 236, 238
  Beispiel 238
Geräte-Gruppe 41, 112
Geräte-Typ 41, 110
Go-Step 28
Graphische Geräte 167
Großraumspeicher 86

Hardware-Adresse 41, 110
Hauptdatenbereich (Prime) für indexsequentielle Dateien 251, 261
Hauptindex für index-sequentielle Dateien 259
Hauptspeicher 86
hexadezimal 32
HIAR, PARM-Unterparameter des Linkage Editor 206
HIARCHY, Steuerkarte des Linkage Editor 212
Hierarchie-Speicher 206, 86, 212
Hilfsspeicher 30

IEBCOPY Dienstprogramm 224
IEBISAM Dienstprogramm 266
IEBUPDTE Dienstprogramm 143, 151
IEFBR14 Pseudoprogramm 69, 188
IEHDASDR Dienstprogramm 171
IEHPROGM Dienstprogramm 231
IEYFORT Programm 207
IN/OUT, LABEL-Unterparameter 198
INCLUDE, Steuerkarte des Linkage Editor 210
INDEX, DSN-Unterparameter 108, 255
Indexbereich für index-sequentielle Dateien 31, 253
  auf einer anderen Einheit 266
  Platzberechnung 260
  SPACE-Parameter 179
Index-sequentielle Dateien 31, 250
  Erzeugung 254
  Hauptdatenbereich 251, 257
  Hauptindex 253, 259
  INDEX im DSN-Parameter 108, 255
  Indexbereich 31, 179, 253, 259, 265
  Reorganisation 266
  Spurindex 251
  Überlaufbereich 252
  unabhängiger Überlaufbereich 253, 261
  Verwendung 262
  Zugriff 253
  Zylinderindex 253
  Zylinderüberlaufbereich 252
Indexstufe, Namensstufe 107
  Erzeugung 234
  Generation Data Group 236
  Löschung 234
Indirekte Adressierung einer direkt organisierten Datei 248
Initialisierungs-Priorität des Jobs 58
Initiator/Terminator 24
Input Stream 24
INSERT, Steuerkarte des Linkage Editor 221

JCL, Job Control Language 17
JCL-Folgekarte 51
JCL-Kartenformate 47
JCL-Kartentypen 19
Job 25
JOB-Karte 53, 79
Job Management 20
Job Scheduler Meldungen 59
JOBLIB DD-Karte 66
  mit DDNAME 159
Jobname, Parameter der JOB-Karte 54
Jobstep 26

Kanal 247
Kanaltrennung 247
Kapazitätsformeln 174
Kartenleser 164
Kartenstanzer 164
Katalogisierte Datei 39, 230
Katalogisierte Verfahren 142, 25, 45
  Beispiel 45
  Erzeugung 143
  nicht erlaubte JCL-Karten 144
  Testen 146
Katalogisierung von Dateien über mehrere Datenträger 123, 241
KEEP, DISP-Unterparameter zum Halten von Dateien 121
Kennsatz 103
KEYLEN, DCB-Unterparameter zur Angabe der Schlüssellänge 184
  bei index-sequentiellen Dateien 251
  im SPACE-Parameter 175
Kluft 33, 192
Konventionelle E/A-Geräte 164, 24

LABEL-Parameter 98, 185
  bei Bändern 193
  bei Plattendateien 185
  EXPDT-Unterparameter 243
  PASSWORD-Unterparameter 243
  RETPD-Unterparameter 243
  Typen bei LABEL 194
Lademodul 28, 204
Laden und Ausführen (Load and Go) 30
Ladepunkt beim Band 192
LET, PARM-Unterparameter des Linkage Editor 206
LIBRARY, Steuerkarte des Linkage Editor 211
Linkage Editor 27, 204
  Beispiel 42, 207
  Fortsetzung der Steuerkarte 210
  JCL 206
  Lage der Steuerkarten 209
  Primäreingabe 205
  Steuerkarten 209, 215, 219
  zusätzliche Eingabequellen 205
  Zusammenfassung 226
LINKEDIT-Programm 206
LIST, PARM-Unterparameter des Linkage Editor 206
Loader 29, 73
LRECL, DCB-Unterparameter zur Angabe der Länge des logischen Satzes 101, 165, 251

Magnetbänder 35, 191
  Schreibverfahren 191
MAP, PARM-Unterparameter des Linkage Editor 206
Member einer untergliederten Datei 91
MFT 21, 24
MOD, DCB-Unterparameter für modifizierte Dateien 117
MODE, DCB-Unterparameter für den Kartenleser 165
Module 25
MSGCLASS, Parameter der JOB-Karte 59
MSGLEVEL, Parameter der JOB-Karte 57, 90
Multiprocessing 21
Multiprogramming 21
MVT 21, 24
MXIG, SPACE-Unterparameter 180

NAME, Steuerkarte des Linkage Editor 215
Name-Feld der JCL-Karten 47
Name-Parameter der JOB-Karte 56
Namensstufe, Indexstufe 107
  Erzeugung 234
  Generation Data Group 236
  Löschung 234

Nationale Zeichen 32
NCAL, PARM-Unterparameter des Linkage Editor 206
NEW, DISP-Unterparameter für neue Dateien 117
Nicht-temporäre Dateien 106
NL, LABEL-Unterparameter für keine Kennsätze 197
Normale Disposition 119
NSL, LABEL-Unterparameter für nicht-standard-Kennsätze 196
NTM, DCB-Unterparameter zur Anforderung eines Hauptindex 260
Nucleus 23
Null-Anweisung 229
NULLFILE-Unterparameter 107, 167

Objektmodul, Objektpaket 27, 204
OLD, DISP-Unterparameter für bestehende Dateien 118
ONLY, COND-Unterparameter 83
OPEN, Makro zum Eröffnen von Dateien 34
Operandenfeld der JCL-Karte 48
Operating System 20
Operationsfeld der JCL-Karte 47
OPTCD, DCB-Unterparameter zur Anforderung zusätzlicher Dienste 245, 247, 259
OS/360 20
OS/360 Systemarten 21
Output Stream 24
OVERLAY, Steuerkarte des Linkage Editor 219
OVFLOW, DSN-Unterparameter 255
OVLY, PARM-Unterparameter des Linkage Editor 206

Paralleles Laden von Datenträgern 114
Parameter der DD-Karte für Karteneingabe 155
Parameter der JOB-Karte und EXEC-Karte 79
Parameter im Operandenfeld 48
PARM-Parameter der EXEC-Karte 71
  Linkage Editor 206
  Loader 73
  unterstellte Werte 72
Partition 23
PASS, DISP-Unterparameter zum Weiterreichen von Dateien 120
PASSWORD, LABEL-Unterparameter für den Dateischutz 243
PCP 21
PGM, Parameter der EXEC-Karte zur Benennung des Programms 64
Platz für das Inhaltsverzeichnis einer untergliederten Datei 178
Platzzuordnung 172
  Primär-Zuordnung 173
  Sekundär-Zuordnung 173
  Anforderung bestimmter Spuren 182
  Beispiel 187
  Blockweise Anforderung 173
  CYL-Unterparameter 175
  index-sequentielle Datei 259
  Platz für das Inhaltsverzeichnis 178
  Platz für Indextabelle 175
  TRK-Unterparameter 179
  Weitervergabe von Platz 184
  Zylinderaufteilung zwischen mehreren Dateien 182
Positionsparameter 48
PRESRES, Glied der SYS1. PARMLIB 127
Primär-Zuordnung 173
PRIME, DSN-Unterparameter 255
Priorität 58, 74
PRIVATE, VOL-Unterparameter 130
private Bibliotheken 27
PROC, Parameter der EXEC-Karte 70
PROC-Anweisung 147
Programmabschnitt 205
Programme 25
  Anwendungsprogramme 21
  Erzeugung 212
  in privaten Bibliotheken 65
  in Systembibliotheken 65
  Name 64
  SYSOUT 163
PRTSP, DCB-Unterparameter für Zeilentransport 164
PRTY, Parameter der JOB-Karte 58
Prüfbitfehler 191
Puffer 34

Qualifizierte Dateinamen 107
Quellenpaket, Quellenmodul 27
Quellensprache 25
Queued-Zugriffstechnik 35

RD, Parameter für den Wiederanlauf 89
in der EXEC-Karte 90
in der JOB-Karte 91
Reader/Interpreter 23
RECFM, DCB-Unterparameter zur Angabe des Satzformates 102
REF, VOL-Unterparameter zur Angabe der Datenträgernummer 128
Regeln für die Kodierung von JCL-Karten 50, 51
Regeln für Parameter der JCL-Karten 49
REGION-Parameter 86
in der EXEC-Karte 87
in der JOB-Karte 88
Region bei MVT 23
Relative Satznummer einer direkt organisierten Datei 249
Relative Spuradresse einer direkt organisierten Datei 249
RELEASE, Steuerkarte für IEHPROGM 236
RENAME, Steuerkarte für IEHPROGM 233
Reorganisation index-sequentieller Dateien 266
Reorganisation untergliederter Dateien 225
RESTART, Parameter der JOB-Karte 91
RETAIN, VOL-Unterparameter zum Halten von Private Datenträgern 131
RETPD, LABEL-Unterparameter zur Angabe eines Aufbewahrungszeitraumes 243
RKP, DCB-Unterparameter zur Angabe der relativen Stellung des Ordnungsbegriffes im Satz 251
RLSE, SPACE-Unterparameter zur Freigabe von Platz 179
ROLL-Parameter
der EXEC-Karte 85
der JOB-Karte 85
Rollout/Rollin Angaben 84
Rotationsverzögerung 37
ROUND, SPACE-Unterparameter zum Aufrunden auf ganze Zylinder 181
Rückbezug-Parameter 43
in der DD-Karte 100
in der EXEC-Karte 69
im Parameter DDNAME 159
im Parameter SUBALLOC 185
Rückkehr-Code 81

Sätze vom Typ Spanned Records 245
Satz 33
fester Länge 33
Folgespureinrichtung 245
index-sequentiell 250
logischer 33
nicht definierter Länge 33
physischer 34
variabler Länge 33
Schließen der Datei 34
Schlüssel 31
Schlüsselworte 48
Schlüsselwortparameter 48
in der EXEC-Karte 70
Schreibring 192
Schutz durch Aufbewahrungszeitraum 243
SCRATCH, Steuerkarte für IEHPROGM 232
Segment 28, 218
Segmentierung des Programmes 218
Sekundär-Zuordnung 173, 176
SEP, UNIT-Unterparameter zum Trennen von Dateien 115
SEP, Parameter für die Kanaltrennung 247
sequentielle Dateiorganisation 30
SER, VOL-Unterparameter zur Angabe der Datenträgernummer 128
SHR, DISP-Unterparameter für gemeinsame Benutzung von Dateien 118
SL, LABEL-Unterparameter für Standard Kennsätze 195
Sonderzeichen 32
Sortierfolge 31
SPACE, Parameter der DD-Karte 172
bei index-sequentiellen Dateien 255
bei SYSOUT Dateien 162
SPLIT, Parameter der DD-Karte zur Zylinderaufteilung 182
Spooling 23
Sprachübersetzer 21
Spuren 37, 170
Magnetband 35, 191
index-sequentiell 251
Spurindex für index-sequentielle Dateien 251
STACK, DCB-Unterparameter zur Auswahl des Ablagefaches beim Stanzer 165
Standard-Blöcke 246
Stanzausgabedatei 161
Status der Datei (DISP) 116
Step 26
Step Restart 90
STEPLIB DD-Karte 67, 213
Stepname der EXEC-Karte 64
Steuerprogramme 20

Streifenspeicher 37, 110, 113, 116, 170
SUBALLOC, Parameter der DD-Karte für Platzanforderung 184
Subroutine 25
Subtasking 25
Suchbewegung (Seek) 37
Suchverzögerung 25
SUL, LABEL-Unterparameter für Standard und Benutzer Kennsätze 196
Symbolische Parameter 146
  Anwendung 151
  Fortsetzung 148
  Kodierung 147
  Löschung 149
  Verknüpfung 148
Synchronisation der Ein-/Ausgabe 35
SYSABEND DD-Karte 76
SYSCHK DD-Karte 94
SYSCTLG, Datei 230
SYSIN DD-Karte 155
SYS1. LINKLIB, Datei 63
SYS1. PARMLIB, Datei 127
SYS1. PROCLIB, Datei 143
SYSUDUMP DD-Karte 76

Task Management 20
TEMPNAME 215
Temporäre Bibliotheken 68
Temporäre Datei 105
TIME, Parameter der EXEC-Karte zur Angabe einer Zeitgrenze 80
TIME, Parameter der JOB-Karte zur Angabe einer Zeitgrenze 81
Time-Sharing 21
Time-Slicing 22
  beim CLASS-Parameter 59
  beim DPRTY-Parameter 75
  beim PRTY-Parameter 59
TRK, SPACE-Unterparameter zur Platzanforderung in Spuren 175
Trommelspeicher 37, 170
TRTCH, DCB-Unterparameter zur Angabe der Art der Magnetbandaufzeichnung 200
TYPRUN, Parameter der JOB-Karte 60

UCS, Parameter der DD-Karte zur Angabe des Universaltypensatzes 165
Überlagerungsbaum 218
Überlagerungsstruktur 28, 217
Überlappte E/A-Operationen 35
Überlaufbereich für index-sequentielle Dateien 252
Unabhängiger Überlaufbereich für index-sequentielle Dateien 252
  Platzberechnung 261
UNCATLG, DISP-Unterparameter zur Entkatalogisierung einer Datei 123
UNCATLG, Steuerkarte für IEHPROGM 234
UNIT, Parameter der DD-Karte zur Angabe des E/A-Gerätes 109, 162
Untergliederte Datei 31
  Inhaltsverzeichnis, Directory 178
  Neuanlage 225
  Platzzuordnung 188
  SPACE-Parameter 179
  Verdichtung 223
Unterparameterliste 48
Unterprogramm 25
Unterprogrammbibliothek 27, 205, 214

Verarbeitungsprogramme 20
Verdichten untergliederter Dateien 223
Verfahrensbibliothek 143
Verfahrensname 69
VERIFY, UCS-Unterparameter 166
Verkettete Kanalprogramme 246
Verknüpfung symbolischer Parameter 149
Verknüpfung von Dateien 31, 136
Verknüpfung von Dateien bei Verwendung von DDNAME 159
Verzögertes Laden eines Datenträgers 114
Verzögertes Wiederanlaufen 88
VOL, Parameter der DD-Karte zur Angabe von Datenträgerinformationen 126
  Anzahl der Datenträger 132
  beliebiger Datenträger 128
  bestimmter Datenträger 128
  Folgenummer 131
  Halten von Datenträgern 131
  Private Datenträger 130
VTOC 171

Wahlfreie Verarbeitung 31
Weg durch den Überlagerungsbaum 219
Wiederanlauf eines Jobs 91
  automatischer Wiederanlauf 88

MSGLEVEL-Besonderheiten 58
Wiederanlauf vom Checkpoint 93
Wiederanlaufangaben 88
Wiederanlaufangaben RD, RESTART 88
Wort 32
Writer 23
Wurzelsegment 218

XREF, PARM-Unterparameter des Linkage Editor 206

Zeichen, alphanumerische 32
nationale Zeichen 32
Sonderzeichen 32
Zentraleinheit 21
Zugeordnete Arbeitsdateien 188
Zugriff index-sequentieller Dateien 253
Zugriffsarm 37
Zugriffstechniken 34
Basic-Zugriffstechnik 35
Queued-Zugriffstechnik 35
Zuordnungsmeldungen 57
Zuschlag, Sekundär-Zuordnung 176
Zylinder auf Speichern mit direktem Zugriff 37, 170
Zylinderindex index-sequentieller Dateien 253
Zylinderüberlaufbereich für index-sequentielle Dateien 252

MSGLEVEL-Besonderheiten 58
Wiederanlauf vom Checkpoint 93
Wiederanlaufangaben 88
Wiederanlaufangaben RD, RESTART 88
Wort 32
Writer 23
Wurzelsegment 218

XREF, PARM-Unterparameter des Linkage Editor 206

Zeichen, alphanumerische 32
nationale Zeichen 32
Sonderzeichen 32
Zentraleinheit 21
Zugeordnete Arbeitsdateien 188
Zugriff index-sequentieller Dateien 253
Zugriffsarm 37
Zugriffstechniken 34
Basic-Zugriffstechnik 35
Queued-Zugriffstechnik 35
Zuordnungsmeldungen 57
Zuschlag, Sekundär-Zuordnung 176
Zylinder auf Speichern mit direktem Zugriff 37, 170
Zylinderindex index-sequentieller Dateien 253
Zylinderüberlaufbereich für index-sequentielle Dateien 252

Der Autor hat Ingenieurwissenschaften an der Universität von Wyoming und Betriebswirtschaft (M.B.A.) an der Universität von Los Angeles (UCLA) studiert. Nach einer Beschäftigung bei der Douglas Air Craft Corporation ist er heute Systemprogrammierer bei der Rand Corporation in Santa Monica (Kalifornien).

Das Buch behandelt die Job Control-Sprache und zeigt in überzeugender Weise, daß diese Sprache der Schlüssel zum Verständnis des Betriebssystems OS ist. System /360, OS und Job Control-Sprache sind eine Ganzheit.

Es ist dem Autor gelungen, gerade diese Ganzheit durch eine Fülle praktischer Beispiele, die in den Gesamtzusammenhang gestellt werden, zu verdeutlichen. Er verliert sich an keiner Stelle in blasse Theorie; für ihn hört Job-Control-Sprache nicht bei irgendeinem ihrer Parameter auf, sondern umfaßt auch den Linkage Editor, indexsequentielle Dateien oder einen Begriff wie Generation Data Groups.

Verfahren der Datenverarbe
R. Oldenbourg Verlag Münche
John Wiley & Sons GmbH Fra

ISBN 3-48